

**UNIVERSIDADE FEDERAL DE SANTA CATARINA**

**Ambiente de comunicação segura de Internet das Coisas com a utilização do  
MQTT e TLS**

**Eduardo de Meireles Koneski**

**Florianópolis - SC**

**2018 / 2**

UNIVERSIDADE FEDERAL DE SANTA CATARINA  
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA  
CURSO DE SISTEMAS DE INFORMAÇÃO

Ambiente de comunicação segura para Internet das Coisas com a utilização  
do MQTT e TLS

Eduardo de Meireles Koneski

Trabalho de conclusão de  
curso apresentado como parte  
dos requisitos para obtenção  
do grau de Bacharel em  
Sistemas de Informação pela  
Universidade Federal de Santa  
Catarina.

Florianópolis - SC

2018 / 2

Eduardo de Meireles Koneski

Ambiente de comunicação segura para Internet das Coisas com a utilização do  
MQTT e TLS

Trabalho de conclusão de curso apresentado como parte dos requisitos para  
obtenção do grau de Bacharel em Sistemas de Informação pela Universidade  
Federal de Santa Catarina.

---

Prof. Dr. Carlos Becker Westphall, INE/UFSC  
Professor Orientador

---

Bel. Leandro Loffi  
Membro da Banca Examinadora, Coorientador

---

Profa. Dr. Carla Merkle Westphall, INE/UFSC  
Membro da Banca Examinadora

---

Me. Hugo Vaz Sampaio  
Membro da Banca Examinadora

## **AGRADECIMENTOS**

A meus pais, pela criação, amor e carinho ministrados, e valores ensinados.

A minha namorada, pelo apoio emocional e incentivo no desenvolvimento do trabalho.

Ao meu orientador Carlos Becker Westphall, pelo auxílio e direcionamento no desenvolvimento deste trabalho, revisão do texto e disponibilidade de tempo e local para o desenvolvimento do trabalho.

Ao meu Coorientador Leandro Loffi, pelo apoio e dedicação empregados a mim e ao trabalho, por disponibilizar tempo e recursos para a viabilidade do trabalho.

**Resumo:** O presente trabalho foca em alguns dos principais aspectos de segurança, integridade, confidencialidade e autenticação. Estes princípios são aplicados ao contexto de Internet das Coisas, que é um paradigma que visa o uso da Internet para interligar dispositivos a rede mundial de computadores, caracterizado pelo uso de um intermediário, como um microcontrolador. A proposta é dividida em duas etapas principais. A primeira foca em receber dados de sensores e enviar a um servidor na nuvem. A segunda etapa consiste em receber os dados da primeira etapa e armazenar. Para atingir requisitos de segurança, os protocolos MQTT com TLS foram utilizados em conjunto para garantir integridade e confidencialidade, e certificados digitais asseguram a autenticação. O presente trabalho realizou testes de validação tanto da transferência correta dos dados, como de aspectos de segurança relevantes referentes ao mesmo.

**Palavras-chave:** Internet das Coisas. Segurança. Agricultura.

**Abstract:** This paper focuses on some of the key aspects of security, integrity, confidentiality and authentication. These principles are applied to the Internet of Things context, which is a paradigm that aims to use the Internet to connect devices to the world wide computer network, characterized by the use of an intermediary, such as a microcontroller. The proposal is divided into two main stages. The first one focuses on receiving sensor data and sending it to a server in the cloud. The second step is to receive the data from the first step and store it. For security requirements, MQTT protocols with TLS were used together to ensure integrity and confidentiality, and digital certificates ensure authentication. The present work carried out validation tests for both the correct transfer of the data, and of the relevant security aspects related to it.

**Keywords:** Internet of Things. Security. Agriculture.

# SUMÁRIO

<b>LISTA DE FIGURAS</b>	<b>8</b>
<b>LISTA DE TABELAS</b>	<b>9</b>
<b>LISTA DE SIGLAS</b>	<b>9</b>
<b>1. INTRODUÇÃO</b>	<b>11</b>
1.1 PROBLEMATIZAÇÃO	11
1.2 JUSTIFICATIVA	12
1.3 SOLUÇÃO PROPOSTA	13
1.4 OBJETIVOS	13
1.4.1 Objetivo geral	13
1.4.2 Objetivos específicos	13
1.5 METODOLOGIA	13
1.5 ORGANIZAÇÃO DO TRABALHO	15
<b>2. FUNDAMENTAÇÃO TEÓRICA</b>	<b>16</b>
2.1. SEGURANÇA DA INFORMAÇÃO	16
2.1.1 Confidencialidade	17
2.1.2 Integridade	17
2.1.3 Autenticação	18
2.2 PROTOCOLOS	19
2.2.1 TCP	20
2.2.2 MQTT	22
2.2.3 TLS/SSL	26
2.3 IOT COM FOG COMPUTING	29
2.3.1 Sensores	30
2.3.2 Gateways	31
<b>3. TRABALHOS CORRELATOS</b>	<b>33</b>
<b>4. DESENVOLVIMENTO</b>	<b>39</b>
4.1 NODEMCU, SENSORES E GATEWAY	40
4.2 BROKER	44
4.3 SUBSCRIBER	45
<b>5. Análise de Resultados</b>	<b>46</b>
<b>6. CONCLUSÃO E TRABALHOS FUTUROS</b>	<b>51</b>
<b>REFERENCIAS</b>	<b>53</b>
<b>APÊNDICE A - CÓDIGO FONTE</b>	<b>56</b>





## LISTA DE FIGURAS

Figura 1 – Fluxo de metodologia	14
Figura 2 – Troca de mensagens MQTT	20
Figura 3 – Passos de um handshake TCP	21
Figura 4 – Cabeçalho TCP	22
Figura 5 – Estrutura MQTT	23
Figura 6 – Cabeçalho fixo de uma mensagem MQTT	24
Figura 7 – Pilha SSL	27
Figura 8 – Handshake TLS	28
Figura 9 – Camadas de um modelo de IoT	30
Figura 10 – Sensor de fluxo de água	31
Figura 11 – NodeMCU	32
Figura 12 – Comparação entre protocolos	33
Figura 13 – Ambiente de Desenvolvimento	39
Figura 14 – Ambiente IoT	40
Figura 15 – Código NodeMCU	42
Figura 16 – Método de envio de dados ao Broker	43
Figura 17 – Arquivo de configuração do Broker	44
Figura 18 – Subscriber MQTT	45
Figura 19 – Console NodeMCU	46
Figura 20 – Saída Subscriber	47
Figura 21 – Mensagens Wireshark	48
Figura 22 – Lista de conjuntos criptográficos	48
Figura 23 – Resumo inválido	49
Figura 24 – Certificado inválido	50

## **LISTA DE TABELAS**

Tabela 1 – Mensagens MQTT	23
Tabela 2 – Trabalhos Correlatos	35

## **LISTA DE SIGLAS**

**3DES** – Triple Data Encryption Standard

**ACK** – Acknowledgement

**AMQP** – *Advanced Message Queuing Protocol*

**CoAP** – *Constrained Application Protocol*

**DES** – Data Encryption Standard

**GPS** – *Global Positioning System*

**GSM** – *Global System for Mobile Communications*

**HMAC** – *Hash-based Message Authentication Code*

**HTTP** – *HyperText Transfer Protocol*

**IoT** – *Internet of Things*

**MAC** – *Mandatory Access Control*

**MQTT** – *Message Queuing Telemetry Transport*

**RSA** – *Rivest-Shamir-Adleman*

**QoS** – *Quality of Service*

**SQL** – *Structured Query Language*

**SSL** – *Secure Sockets Layer*

**SYN** – *Synchronize*

**TCP** – *Transmission Control Protocol*

**TLS** – *Transport Layer Security*

# 1. INTRODUÇÃO

A utilização de ferramentas para auxílio na agricultura não é nova, e avanços tem sido feito a esse respeito com o uso da tecnologia. Desde tratores para acelerar a coleta de frutas e sementes, até o despejo de pesticidas em lavouras, a tecnologia tem se mostrado presente e essencial na competitividade do negócio.

A tecnologia tem auxiliado em diversos setores produtivos, em organizações de pequeno a grande porte. O uso eficaz da tecnologia ajuda a melhorar desde o processo de negócio, quanto a redução de custos, maximização de lucros ou redução de perdas. Nesse contexto, o uso de tecnologia no setor agrícola pode representar benefícios tanto do ponto de vista comercial, quanto produtivo, pois age em diversos setores da indústria.

Neste trabalho, é apresentado como a tecnologia pode ser utilizada para o fim de auxiliar no monitoramento de sensores, podendo ser útil nas etapas supracitada, e mantendo as informações protegidas.

## 1.1 PROBLEMATIZAÇÃO

Dentre os variados cenários com o uso de dispositivos *Internet of Things* (IoT), podem ser tanto em uma *smart city* como também em ambientes agrícolas. Focando na segunda possibilidade, os dispositivos IoT podem ser bastante importantes ao especialista agrícola que deseja melhorar a qualidade das lavouras. O principal problema apresentado em ambientes agrícolas segundo especialistas é como é possível monitorar áreas que são pulverizadas, mas mesmo assim surgem problemas com doenças e inços de erva daninhas.

Por uma perspectiva tecnológica, os dados que podem ser coletados em uma área agrícola são enviados a um servidor da Internet, no qual, outros usuários

podem ter acesso e alterar valores de medição coletados pelos sensores por meio de interceptações dos dados.

## 1.2 JUSTIFICATIVA

Stallings (2008) cita duas formas de ameaças a sistemas - podendo ser de informação ou serviços - o autor segue explicando que ambos são programas escritos com a finalidade de explorar vulnerabilidades sobre os respectivos domínios. Ataques a informação são caracterizados pelo acesso indevido ou até mesmo alteração de dados sem a devida permissão, como por exemplo, ataque de replay, ou homem no meio. Já ataques de serviço visam comprometer o acesso de usuários a determinados sistemas ou serviços, seja pelo roubo de informação dos serviços, ou pela privação de acesso, ataque conhecido como negação de serviço.

Daisy Ridley (2017) insere IoT no contexto da agricultura e pecuária, projetando crescimento significativo na área de agricultura inteligente, devido a adesão da IoT no ramo, impactando diretamente na minimização de custos e melhores resultados, como menor consumo de água e quantidade de perdas. A autora segue explicando os riscos e variáveis do setor, como variação na precipitação e qualidade do solo, apontando para a necessidade de medições precisas e em tempo real de tais condições e utilizá-las na tomada de decisão.

Problemas neste aspecto podem acarretar grandes perdas ao agricultor por não ter um dispositivo que é preciso e informe dados corretos. Através da segurança da informação pode-se ter uma garantia de tráfego de dados entre os dispositivos, assim o dispositivo final terá os dados corretos.

## 1.3 SOLUÇÃO PROPOSTA

Para solucionar o problema de insegurança no tráfego de dados na comunicação entre os dispositivos coletores de informações de vazão de fungicida e a nuvem, é proposto a solução de utilização do *Message Queuing Telemetry Transport* (MQTT) juntamente com o *Transport Layer Security* (TLS)/*Secure Sockets Layer* (SSL), que usa o certificado digital para fazer a comunicação entre as partes.

## 1.4 OBJETIVOS

### 1.4.1 Objetivo geral

O objetivo geral deste trabalho é o monitoramento através de sensores instalados em uma rede de IoT, que visam a segurança das informações transmitidas pela comunicação da rede.

### 1.4.2 Objetivos específicos

Os objetivos específicos deste trabalho são:

- Montar ambiente IoT;
- Aplicar MQTT com TLS no Ambiente IoT;
- Monitorar o Sistema de vazão de fungicida em plantações;
- Verificar integridade dos dados recebidos pelos dispositivos IoT;

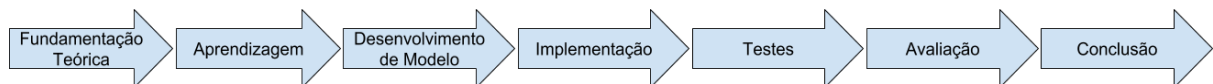
## 1.5 METODOLOGIA

Tendo problemas e objetivos definidos, foi adotada uma estratégia para obter o resultado esperado, conforme a figura 1, que consiste nos seguintes passos:

- Revisão da Literatura;
- Aprendizagem dos temas que estão vinculados a uma rede IoT;
- Desenvolvimento de um modelo de monitoramento em ambientes agrícola;

- Projetar um ambiente com sensores para enviar dados utilizando o protocolo MQTT com TLS/SSL;
- Testar o protótipo em ambiente real;
- Avaliar o modelo proposto;
- Escrever os resultados do trabalho.

Figura 1: Fluxo de metodologia.



Fonte: Autor, 2018.

Os trabalhos anteriores foram os principais pontos de partidas para a forma de obter informação e referencial teórico de autores para o desenvolvimento do trabalho. Os principais temas de pesquisa por referências foram MQTT, NodeMCU, TLS/SSL e segurança, abrangendo também combinações das palavras chave citadas.

Para a realização do trabalho, foi necessário o aprendizado dos temas descritos no referencial teórico. Teve-se como principais temas de pesquisa as redes IoT, Fog Computing e sensores. Ainda foram utilizados outros temas específicos para a montagem do ambiente e implementação do código nos dispositivos IoT. Ou seja, foi aprendido a linguagem do Arduino, C++, Python e *Structured Query Language* (SQL) para o desenvolvimento posterior de um ambiente real.

Em relação ao desenvolvimento do modelo de monitoramento em ambiente agrícola, foi utilizado o auxílio de um especialista na área de agronomia para melhor adequar em um ambiente real o modelo. O modelo partiu do princípio do interesse de monitorar áreas agrícolas que são pulverizadas.

Sensores sobre a plataforma de desenvolvimento NodeMCU foram utilizados, os dados obtidos são então enviados para um servidor na nuvem utilizando o protocolo MQTT na camada de transporte, e para a segurança foi utilizado o protocolo TLS/SSL com certificados.

Com o protótipo finalizado, testes foram feitos sobre ele, em um ambiente simulado. Os testes contemplam as medições dos sensores, o transporte dos dados ao servidor, e o protocolo de segurança implantado.

Com os dados coletados, a avaliação do modelo é feita verificando se os dados enviados são os mesmos recebidos.

Por fim é escrito os resultados adquiridos com o modelo proposto, levando em consideração a metodologia aqui descrita.

## 1.5 ORGANIZAÇÃO DO TRABALHO

Primeiramente será apresentada a fundamentação teórica, com os conceitos necessários para a o desenvolvimento do trabalho. Em seguida são apresentados os trabalhos correlatos, explicando brevemente e comparando ao presente trabalho. Na sequência é explicado o processo de desenvolvimento do trabalho realizado, explicando os passos executados, e os testes que foram feitos sob o ambiente pronto. Por fim são apresentadas as conclusões e trabalhos futuros.



## 2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão tratados todos os conceitos envolvidos no trabalho, abrangendo princípios de segurança, Internet das Coisas e protocolos utilizados.

Inicialmente são tratados temas relacionados a segurança da informação, passando por princípios básicos de segurança, dando então maior enfoque em explicar os conceitos de segurança relevantes para o presente trabalho.

Em seguida, são abordados os fundamentos de protocolos, com conceitos gerais dos mesmos, e dando maior foco aos protocolos específicos do trabalho. No presente trabalho, são explorados os protocolos *Transmission Control Protocol* (TCP), MQTT e TLS/SSL.

### 2.1. SEGURANÇA DA INFORMAÇÃO

Segurança da informação é um conjunto de ferramentas, políticas, conceitos, ações e práticas que podem ser utilizadas em conjunto com a finalidade de proteger um ambiente, seja ele organizacional ou pessoal, podendo agir sob dispositivos, aplicações, serviços, entre outros. (VON SOLMS, VAN NIEKERK, 2013). Sistemas de informação contém diversos tipos de serviços, existindo assim diversos pontos para existir vulnerabilidades, as quais a segurança da informação busca evitar por meio de seus diversos artifícios para o fim de manter o sistema seguro.

Do Espírito Santo (2011) descreve segurança da informação como sendo um mecanismo essencial para as organizações, que tem as finalidades de proteger a informação e também como forma de gerenciamento. Qualquer organização possui algum tipo de sistema conectado a uma rede ou a Internet, dessa forma, é necessário algum método de mantê-lo seguro, diminuindo riscos a informação sensível e possíveis prejuízos decorrentes de falhas de segurança.

Padrões internacionais definem os principais aspectos da segurança da informação, entre essas, destaca-se: Confidencialidade e integridade. Adicionalmente, podem ser citadas a autenticidade e o não-repúdio, como outros aspectos importantes na segurança da informação. Sistemas de segurança relevantes trabalham com essas premissas para obter os melhores resultados, sendo esses essenciais para manter organizações e usuários seguros.

### **2.1.1 Confidencialidade**

O princípio da confidencialidade, podendo também ser chamado de confiabilidade, descreve que somente usuários autorizados terão acesso à informação. Segundo Stallings (2008), aspectos relevantes da confidencialidade são o sigilo de mensagens trocadas entre emissor e receptor, vedando a terceiros o acesso a tais mensagens.

Uma das melhores formas de se garantir a confidencialidade é por meio da criptografia das mensagens trocadas entre as partes envolvidas, emissor e receptor. (STALLINGS, 2008). A criptografia é uma técnica que embaralha as mensagens enviadas pela rede, por meio de uma ou mais senhas, para garantir a confidencialidade e garantir que agentes externos não sejam capazes de obter acesso ao conteúdo de mensagens trocadas (STALLINGS, 2008).

### **2.1.2 Integridade**

Integridade é a capacidade de um sistema de qualquer natureza de prevenir, detectar e impedir modificação ou corrupção de conteúdo estando ou não em trânsito, independente da natureza, seja de forma acidental ou intencional (STALLINGS, 2008). Stallings (2008) descreve integridade como a garantia de que uma mensagem enviada será a mesma mensagem entregue ao destinatário.

Existem algumas técnicas que validam a paridade entre mensagens enviadas e recebidas pelas partes, a mais conhecida delas é a do cálculo de hash, um resumo comparado por ambas as partes, que valida mensagens enviadas, por meio de cálculos feitos sobre a mensagem original. Para garantir uma comunicação com integridade entre duas partes, o emissor deve primeiramente calcular o *hash* da mensagem que deseja enviar, então enviar o conteúdo da mensagem juntamente com o seu *hash*, que após o receptor receber, deverá realizar o procedimento supracitado, e comparar ao *hash* enviado pela outra parte por fim, devem apresentar o mesmo conteúdo.

### **2.1.3 Autenticação**

Filho (2009) descreve a autenticação em sistemas como sendo um processo com a finalidade de verificar a identidade de um indivíduo a um sistema ou recurso. O autor segue explicando sobre os modos de autenticação, podendo ser definido por algo que o usuário é, com artifícios de biometria, leitura de retina, reconhecimento facial, entre outros. Outro modo é caracterizado por algo que o usuário possui, como *tokens* de autenticação, *smart cards*, etc. Algo que o usuário conhece, como senhas fixas ou senhas de uso único. E finalmente, onde o usuário está, como a utilização de uma máquina específica.

Ainda segundo Filho (2009), tokens de autenticação frequentemente possuem algum tipo de certificado digital, contendo um par de chaves, impedindo que a chave privada do usuário seja exposta, contando com algum tipo de criptografia, como *Rivest-Shamir-Adleman* (RSA), *Data Encryption Standard* (DES), ou *Triple Data Encryption Standard* (3DES). Nesse modelo de autenticação, chaves públicas e privadas são objetos de comparação entre cliente e servidor, bem como

seus respectivos certificados digitais, para garantir que as partes são de fato quem dizem ser.

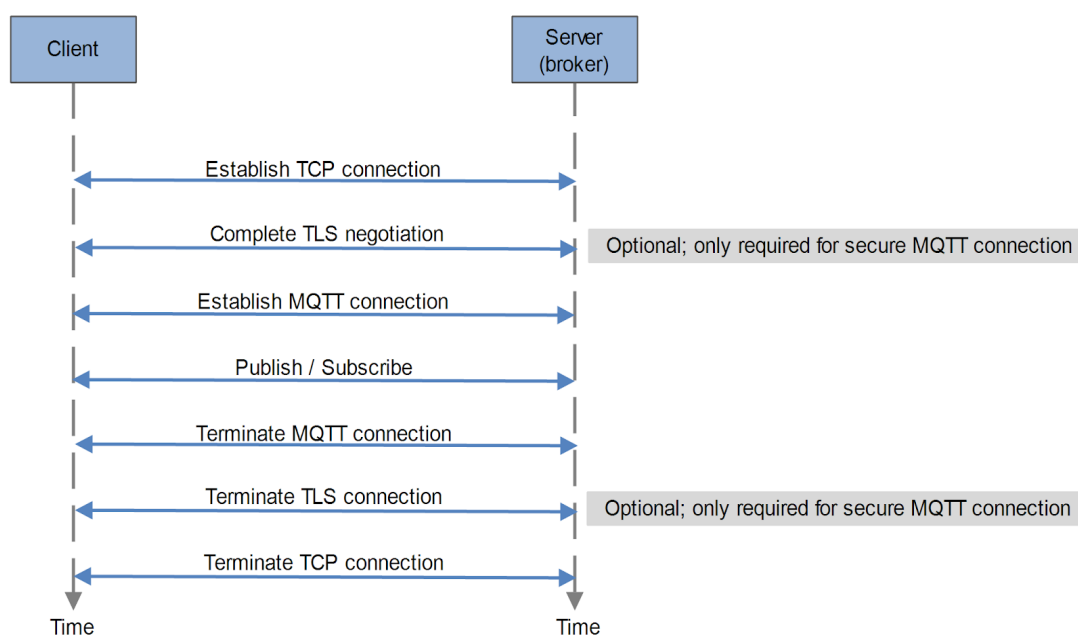
## 2.2 PROTOCOLOS

Protocolos de comunicação podem ser definidos como um conjunto de regras, convenções, padrões e procedimentos para o fim de padronizar e realizar a comunicação entre diferentes equipamentos. Dessa forma, protocolos são padrões de comunicação para possibilitar a comunicação entre diferentes dispositivos, como desktops, celulares, ou qualquer outro dispositivo capaz de se conectar a rede.

Castelucci (2011) considera essencial que existe um padrão de comunicação entre dispositivos conectados a uma rede. Ao longo dos anos, diferentes protocolos foram criados para diferentes fins, tornando cada um deles específico a solucionar algum problema a que se propõe.

No trabalho proposto, os protocolos que foram utilizados são TCP e MQTT para transporte, uma vez que o protocolo TCP é amplamente difundido, e o protocolo MQTT, que opera sobre o TCP, é uma alternativa viável de comunicação entre dispositivos IoT, contando também com o protocolo TLS/SSL para suprir requisitos de segurança. A figura 2 mostra a ordem em que os protocolos atuam sobre o ambiente proposto.

Figura 2: Troca de mensagens MQTT.



Fonte:

[https://doc.hcc-embedded.com/download/attachments/20644367/MQTT%20Sequence.gif?version=1](https://doc.hcc-embedded.com/download/attachments/20644367/MQTT%20Sequence.gif?version=1&modificationDate=1489055434000&api=v2)  
&modificationDate=1489055434000&api=v2. Acessado em 18/10/2018.

## 2.2.1 TCP

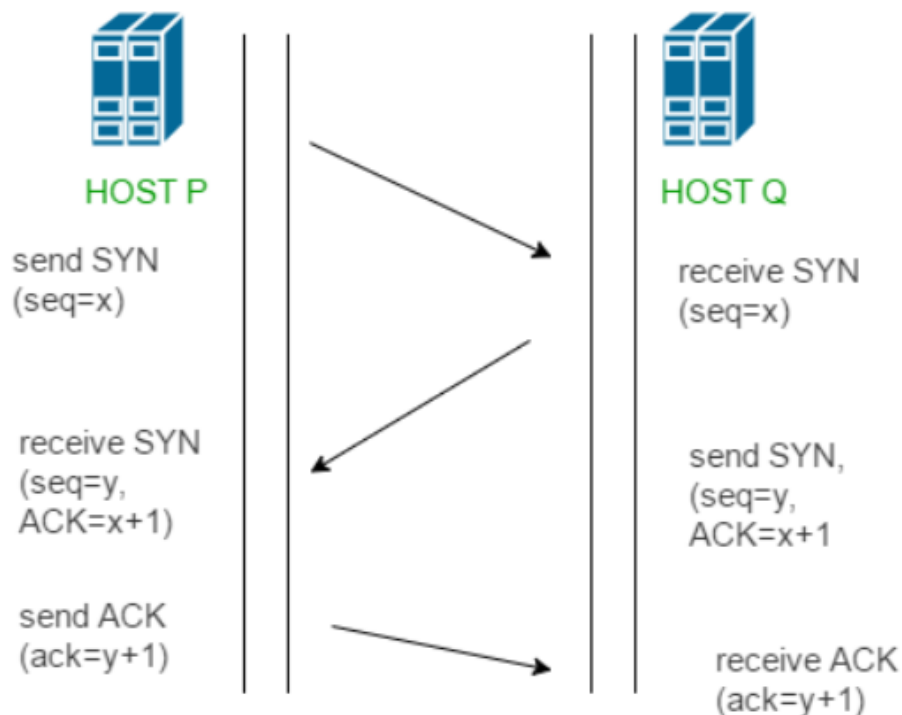
O protocolo de controle de transmissão, ou *Transmission Control Protocol* (TCP), é o protocolo mais utilizado na Internet, pois verifica se os dados são enviados de forma correta, na ordem correta e sem erros (STALLINGS, LAWRIE, 2013). O protocolo foi proposto na década de 70, como forma de comunicação entre máquinas em uma rede pelo meio de troca de pacotes, que eventualmente se tornou o protocolo da Internet.

Gugelmin (2014) descreve que o funcionamento desse protocolo segue o modelo cliente-servidor. O protocolo se inicia pela requisição do cliente ao servidor, requerendo dados dele, e então o servidor retorna os dados para o cliente, quando o processo é concluído, o cliente por sua vez faz uma checagem de erros nos pacotes, por meio de um sistema de numeração adotado pelo protocolo para

detectar fragmentos corrompidos, omissos ou desordenados. Caso a informação esteja correta, o cliente confirma que os dados foram transferidos com sucesso, caso contrário, o servidor enviará os dados novamente até que o cliente de a confirmação de que os dados estão corretos.

Para iniciar uma conexão, é necessário que haja um acordo entre as duas partes, conhecido como *handshake* de três vias, que verifica se ambas as partes estão prontas para o envio de dados (STALLINGS, LAWRIE, 2013). Para realizar o *handshake* representado na figura 3, o cliente deve escutar a própria porta, para então receber uma escuta ativa de um cliente, conhecido como *Synchronize* (SYN), enviando um número aleatório  $x$ . O servidor então retorna  $x+1$  ao cliente, conhecido como *Acknowledgement* (ACK), e um outro número aleatório  $y$ . Por último, o cliente recebe  $y+1$ , validando assim a conexão entre as partes.

Figura 3: Passos de um handshake TCP.

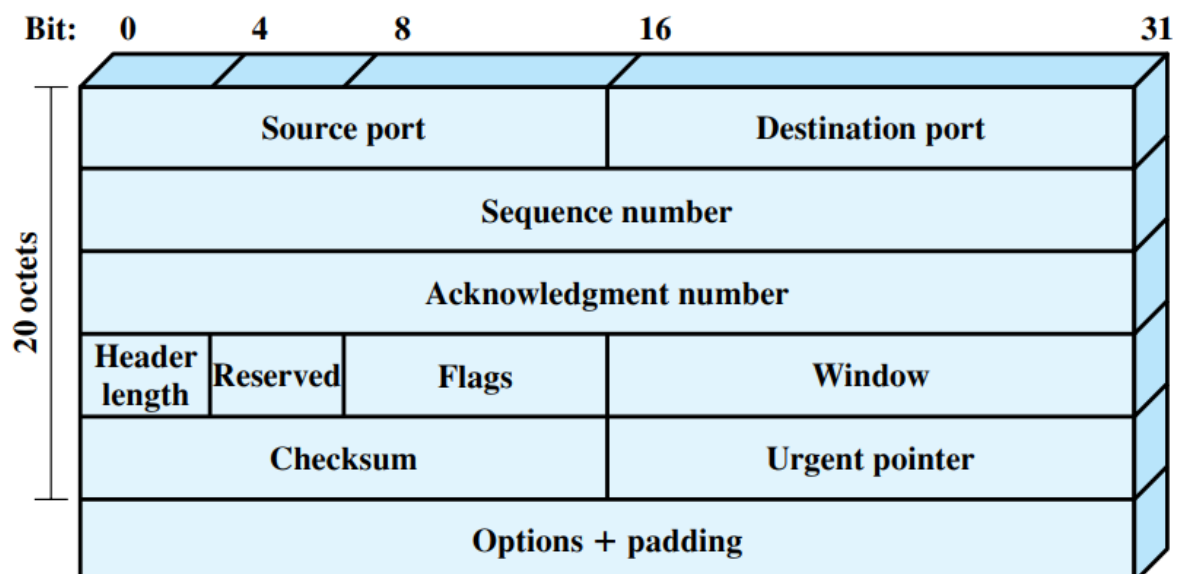


Fonte: <https://cdncontribute.geeksforgeeks.org/wp-content/uploads/TCP-connection-1.png>. Acessado

em 18/10/2018.

Com o *handshake* finalizado, é possível que os hosts transfiram blocos de dados entre si, esses blocos de tamanho variável são organizados por informações (STALLINGS, LAWRIE, 2013). A figura 4 representa o cabeçalho TCP, que contém um conjunto de 4 *bytes*, cada um contendo uma quantidade de bits reservada para cada parâmetro, como as portas fonte de envio e destino, número de sequência, *acknowledgment* e janela são os parâmetros de controle da ordem dos blocos, enquanto o *checksum* é encarregado de verificar erros no segmento.

Figura 4: Cabeçalho TCP.



Fonte: Stallings, 2008.

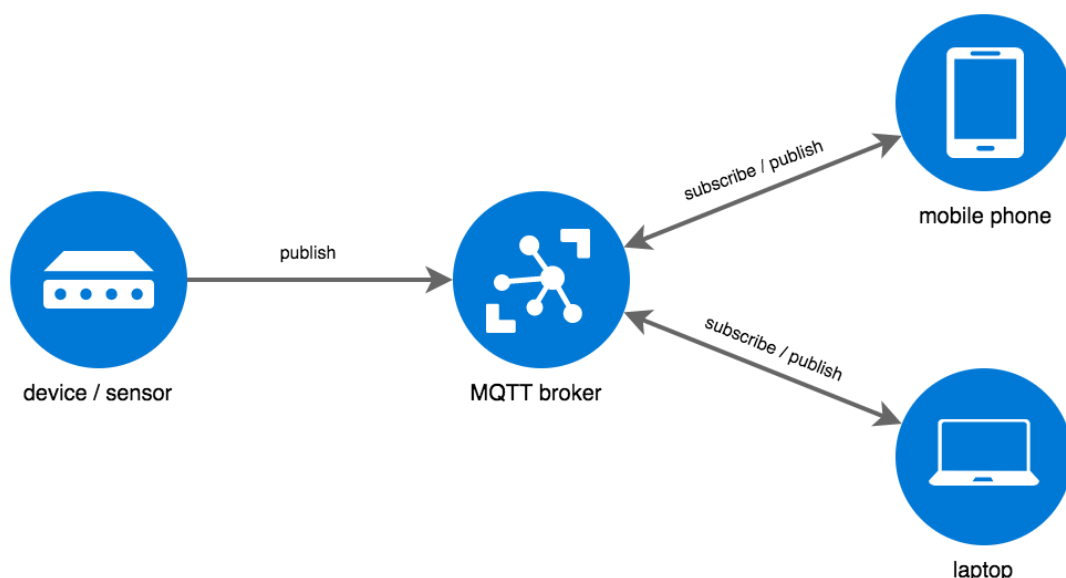
### 2.2.2 MQTT

O protocolo da camada de aplicação *Message Queuing Telemetry Transport* (MQTT), foi criado na década de 90, com a premissa de ser simples, leve e de fácil implementação (OASIS, 2014). O autor ainda cita que a forma com que as mensagens são trocadas favorecem o seu uso no ambiente de Internet das Coisas e em ambientes restritivos em termos de recursos e largura de banda.

Ainda segundo Oasis (2014), a estrutura do protocolo MQTT utiliza o padrão de comunicação cliente-servidor, onde um ou mais clientes se conectam a um servidor, ou *broker*. Clientes podem enviar dados ao *broker* por meio de publicação, ou receber dados por meio de subscrição, informando um tópico, que é uma forma de organização de informação por parte do *broker*, que por sua vez distribui dados recebidos a qualquer *subscriber* do mesmo tópico, utilizando o protocolo TCP para o transporte entre os pontos.

Uma estrutura representada na figura 5 mostra diferentes dispositivos enviando e recebendo dados por meio do protocolo MQTT. O sensor tem o papel apenas de publicar dados para o *broker* que por sua vez os encaminha aos outros dispositivos *subscribers*.

Figura 5: Estrutura MQTT.



Fonte: <https://1sheeld.com/wp-content/uploads/2018/07/Pure-javascript-MQTT-broker.png>.

Acessado em: 18/10/2018

Oasis (2014) explicam sobre o cabeçalho das mensagens que transitam pelo protocolo MQTT, tendo um tamanho fixo de dois bytes, com parâmetros específicos.



Representado na figura 6, o cabeçalho fixo das mensagens MQTT é composto por parâmetros que indicam o tipo da mensagem, indicador de mensagem duplicada, marcador de qualidade do serviço, marcador de retenção e a largura restante representando o espaço reservado ao restante da mensagem, contendo o cabeçalho variável e o conteúdo da mensagem, chamado de payload.

Figura 6: Cabeçalho fixo de uma mensagem MQTT.

bit	7	6	5	4	3	2	1	0
Byte 1	Tipo da Mensagem				Flag DUP	Nível QoS		RETAIN
Byte 2	Largura restante							

Fonte: Martins, Zem, 2015.

Os tipos de mensagens MQTT são os mais diversos, a maior parte deles contendo uma mensagem *acknowledgment* correspondente. A tabela 1 apresenta as mensagens existentes no protocolo MQTT, contendo seu nome, valor, sentido e descrição.

Tabela 1: Mensagens MQTT.

Name	Value	Direction of Flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Client request to connect to Server
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment

PUBREC	5	Client to Server or Server to Client	Publish received (assured delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (assured delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client to Server	Client subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgment
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server	Client is disconnecting
Reserved	15	Forbidden	Reserved

Fonte: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>

O restante da mensagem é composto pelo cabeçalho variável, contendo nome e versão do protocolo nas mensagens CONNECT, além de algumas diretivas para a conexão entre cliente e *broker*. O restante do espaço reservado está destinado ao *payload*, que pode conter diversos tipos de dados, a depender do tipo da mensagem que está sendo transmitida. A mensagem CONNECT por exemplo, possui no seu *payload* o ID do cliente, enquanto a mensagem PUBLISH contém o valor a ser enviado ao *broker*. (OASIS, 2014).

Os níveis de *Quality of Service* (QoS) definem como o *broker* deve se comportar a cada requisição, podendo ser 00 para nenhuma ação, 01 para retornar ACK, 10 para garantir entrega, e 11 é reservado ao protocolo (OASIS, 2014).

### 2.2.3 TLS/SSL

O protocolo TCP executa o trabalho de garantir a transferência de dados de um ponto a outro, mas no mesmo não é implementado qualquer tipo de segurança, enviando dados não criptografados pela rede entre as partes envolvidas, comprometendo a segurança na comunicação e potencialmente violando os princípios de segurança anteriormente descritos (STALLINGS, 2008).

Stallings (2008) afirma que o protocolo *Transport Layer Security* (TLS), também chamado de *Secure Sockets Layer* (SSL) age sobre o TCP como uma forma de troca de dados eficiente e segura para ambas as partes, de um ponto a outro, garantindo a segurança na comunicação. O protocolo foi criado na década de 90, recebendo sucessivas alterações e adaptações para acompanhar os padrões de segurança exigidos para garantir a segurança dos dados trafegados, estando atualmente na sua versão 1.3 (STALLINGS, 2008).

O protocolo SSL é dividido em alguns protocolos que operam em camadas superiores (STALLINGS, LAWRIE, 2013). A figura 7 mostra o protocolo SSL atuando sobre o protocolo TCP, cuja função é fragmentar os dados, podendo seguir para um dos seguintes protocolos listados a seguir:

- Protocolo de estabelecimento de sessão SSL, responsável pelo *handshake* entre os pontos.
- Protocolo de mudança de especificação de cifra SSL, que indica alteração nos parâmetros na conexão.

- Protocolo de alerta SSL, responsável por alertar sobre problemas no protocolo, e caso o alerta seja fatal, a conexão é imediatamente interrompida.

Figura 7: Pilha SSL.



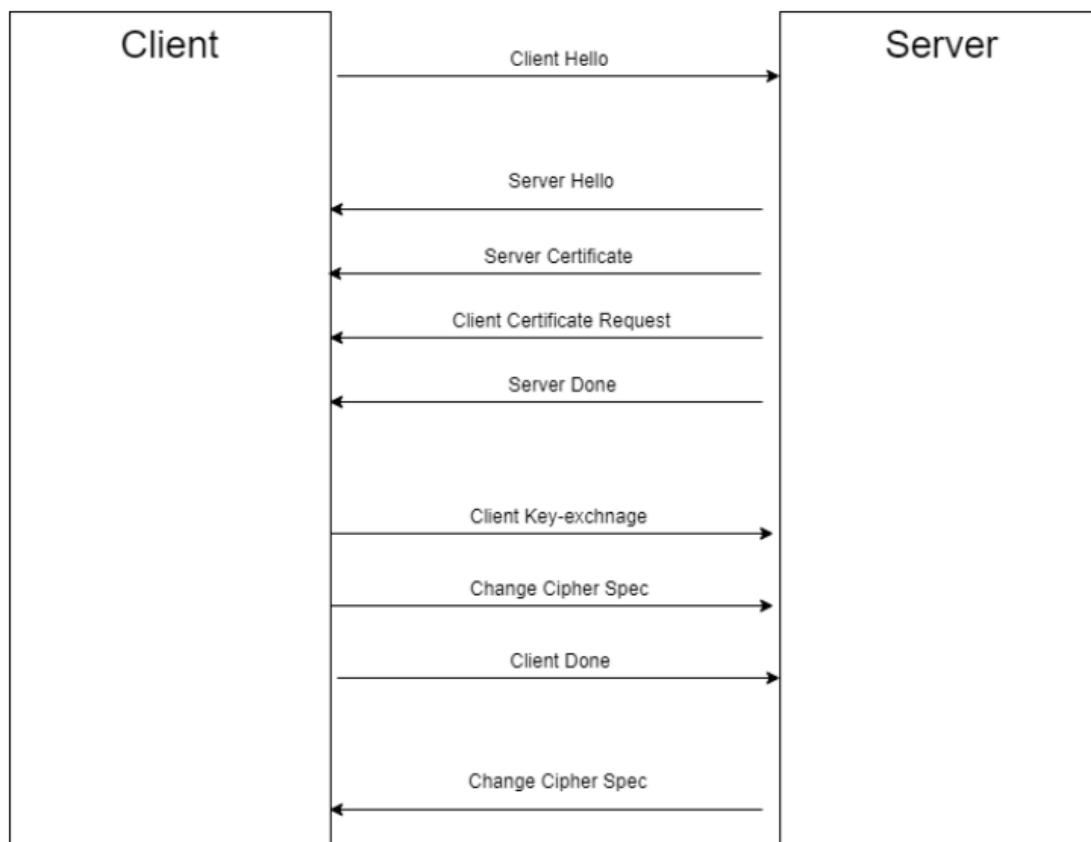
Fonte: Stallings, 2008.

O protocolo TLS está fortemente relacionado ao SSL, sendo um sucessor dele (STALLINGS, LAWRIE 2013). O TLS possui algumas vantagens em relação ao SSL, como a utilização de diferentes portas e algoritmos criptográficos para atuação, utilizando Hash-based Message Authentication Code (HMAC), ao invés do mandatory access control (MAC) presente no SSL.

Antes de trocar mensagens criptografadas, é necessário que haver um *handshake* entre as partes, para isso, mensagens são enviadas para estabelecer a conexão segura (STALLINGS, LAWRIE 2013). A figura 8 representa o *handshake* do protocolo TLS, com todos os passos necessários para que a conexão seja efetuada de forma segura.

A primeira mensagem é o *Client Hello*, enviada do cliente ao servidor, enviando uma lista de algoritmos criptográficos suportados pelo mesmo, juntamente com um número aleatório utilizado posteriormente.

Figura 8: Handshake TLS.



Fonte: CARLSSON, ERIKSSON, 2018.

Em seguida o servidor responde com um *server hello*, uma mensagem que mostra o algoritmo escolhido dentre a lista enviada no *client hello*, juntamente com um ID de sessão, uma sequência de *bytes* aleatória e o certificado digital do servidor ao cliente, e uma mensagem *server done*, indicando ao cliente que inicie a troca de chaves.

Após a verificação do certificado digital do servidor pelo cliente, a troca de chaves é iniciada pelo cliente, que envia a sequência de *bytes* aleatória de volta ao servidor, que será utilizada para cifrar as mensagens, enviando assim um aviso de definição de nova especificação de chaves, terminando com uma mensagem *client done*, a primeira mensagem utilizando a chave secreta estabelecida. A partir daí,

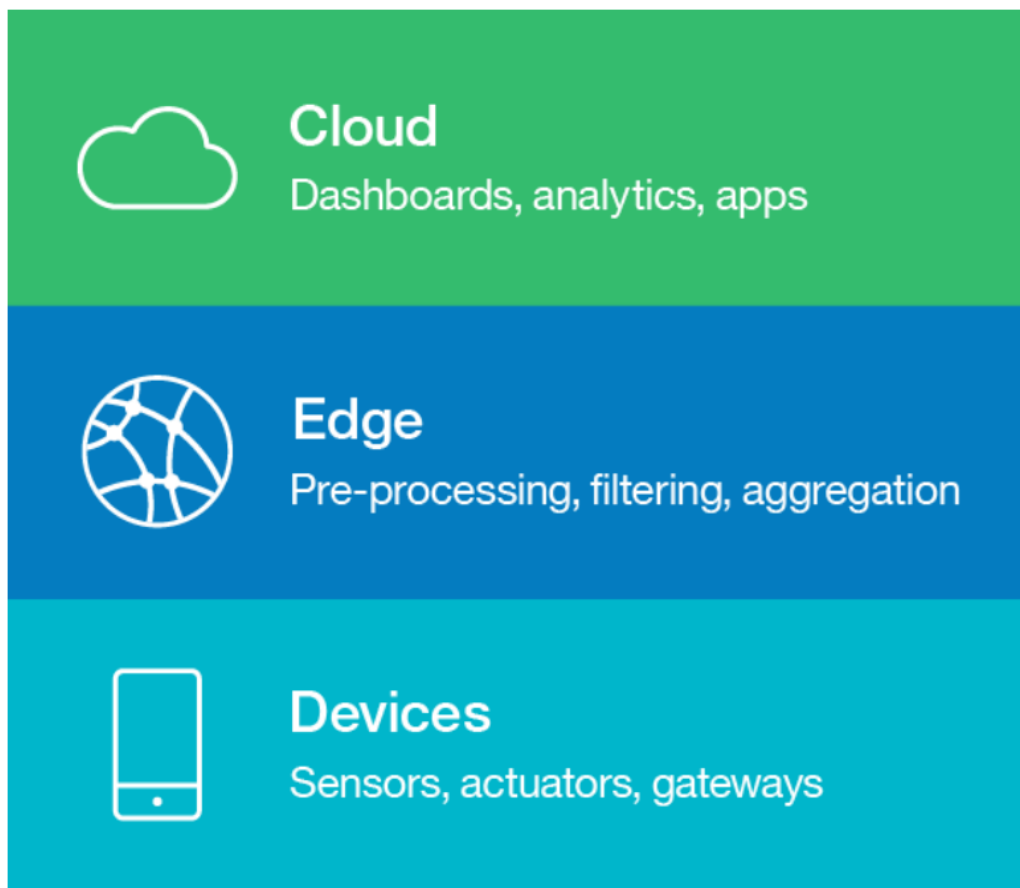
novas especificações podem ser feitas entre as partes, com a utilização de novas mensagens *Change Cipher Spec*.

## 2.3 IOT COM FOG COMPUTING

O acesso a Internet não mais se limita a *desktops* ou servidores, uma variedade de dispositivos possuem acesso a recursos que utilizam a Internet das Coisas como um ambiente no qual objetos são integrados, transmitindo dados entre si e com o ambiente externo (Internet), onde serviços podem ter acesso a dados dos objetos (dispositivos) de qualquer lugar, por meio da Internet, tornando os objetos (coisas) inteligentes, e possibilitando uma ponte entre esses objetos e a Internet.

No contexto de computação na nuvem e Internet das Coisas, problemas começaram a surgir, principalmente no que se refere a quantidade de dados trafegando pela rede, consumindo banda pela Internet de um ponto a outro. A computação em nevoeiro é um modelo que adiciona um ponto entre os objetos do modelo de Internet das Coisas e a nuvem, reduzindo a latência e a quantidade de dados enviados para a nuvem, centralizando armazenamento e processamento sem a necessidade de intervenção externa (BONOMI, 2012). A figura 9 ilustra um modelo hipotético de IoT utilizando uma camada intermediária entre dispositivos IoT e a nuvem.

Figura 9: Camadas de um modelo de sistema IoT.



Fonte: Simplifique o desenvolvimento de soluções de IoT com arquiteturas de IoT. Disponível em <https://www.ibm.com/developerworks/br/library/iot-lp201-iot-architectures/index.html>. Acessado em 16/10/2018.

### 2.3.1 Sensores

No modelo de Internet das Coisas, sensores são dispositivos capazes de sentir e mensurar condições do ambiente onde se encontram fisicamente, sendo eles autônomos, ou dependentes de ação do usuário, transmitindo esses dados para as partes interessadas (HALLER, KARNOUSKOS, SCHROTH, 2008). Temperatura, umidade, velocidade, geolocalização são algumas das condições ambientais que podem ser mensuradas por sensores, que podem ser utilizadas por outros dispositivos, que recebem dados de um ou mais sensores para resolver um problema.

No presente trabalho, coletando dados e enviando a parte interessada, processando os mesmos como forma de resolver o problema proposto. A figura 10 representa um sensor de vazão de água.

Figura 10: Sensor de Fluxo de água.



Fonte: Sistema De Monitoramento De Consumo De Água Utilizando o Protocolo De Comunicação MQTT. Disponível em <http://repositorio.ufu.br/bitstream/123456789/21883/1/SistemaMonitoramentoConsumo.pdf>. Acessado em 16/10/2018.

### 2.3.2 Gateways

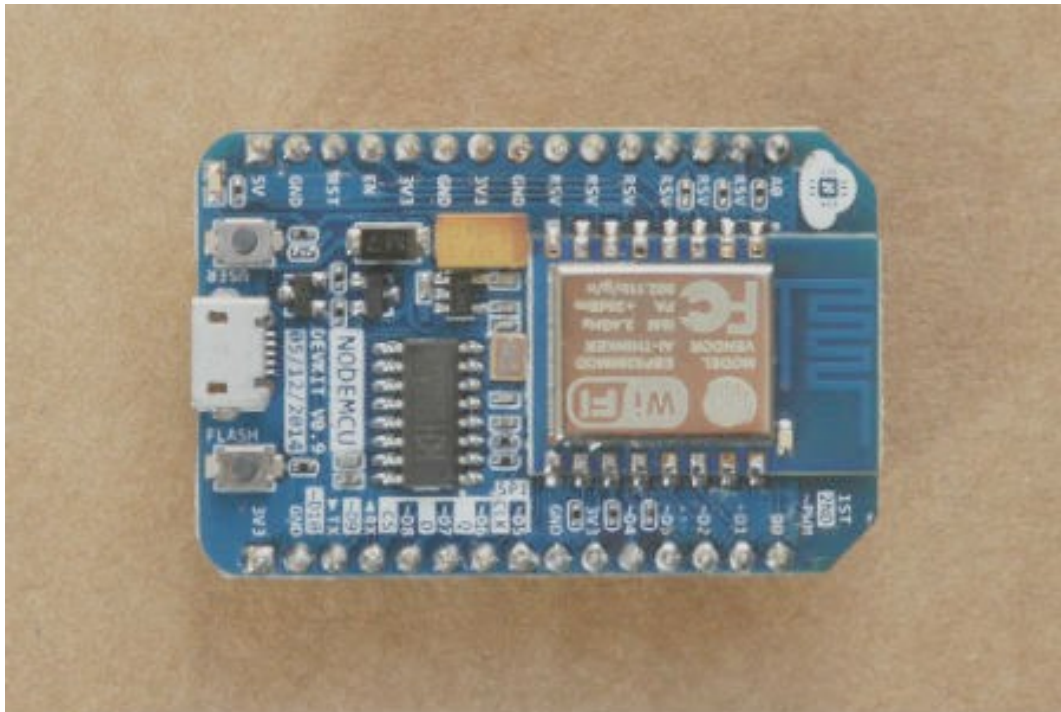
No modelo de *fog computing*, *gateways* recebem os dados dos sensores e executam o pré-processamento e armazenamento dos mesmos quando necessário, sendo também responsável pelo envio de dados a nuvem (UCKELMANN, HARRISON e MICHAHELLES, 2011). Sensores enviam dados ao gateway, para serem posteriormente enviados a nuvem, tornando os dados disponíveis ao usuário para ação posterior.

No presente trabalho, o microcontrolador NodeMCU, representado na figura 11, exerce o papel de *Gateway* no ambiente proposto, recebendo os dados



coletados e por meio de uma conexão WiFi, enviando de forma segura ao servidor para posterior manipulação.

Figura 11: NodeMCU.



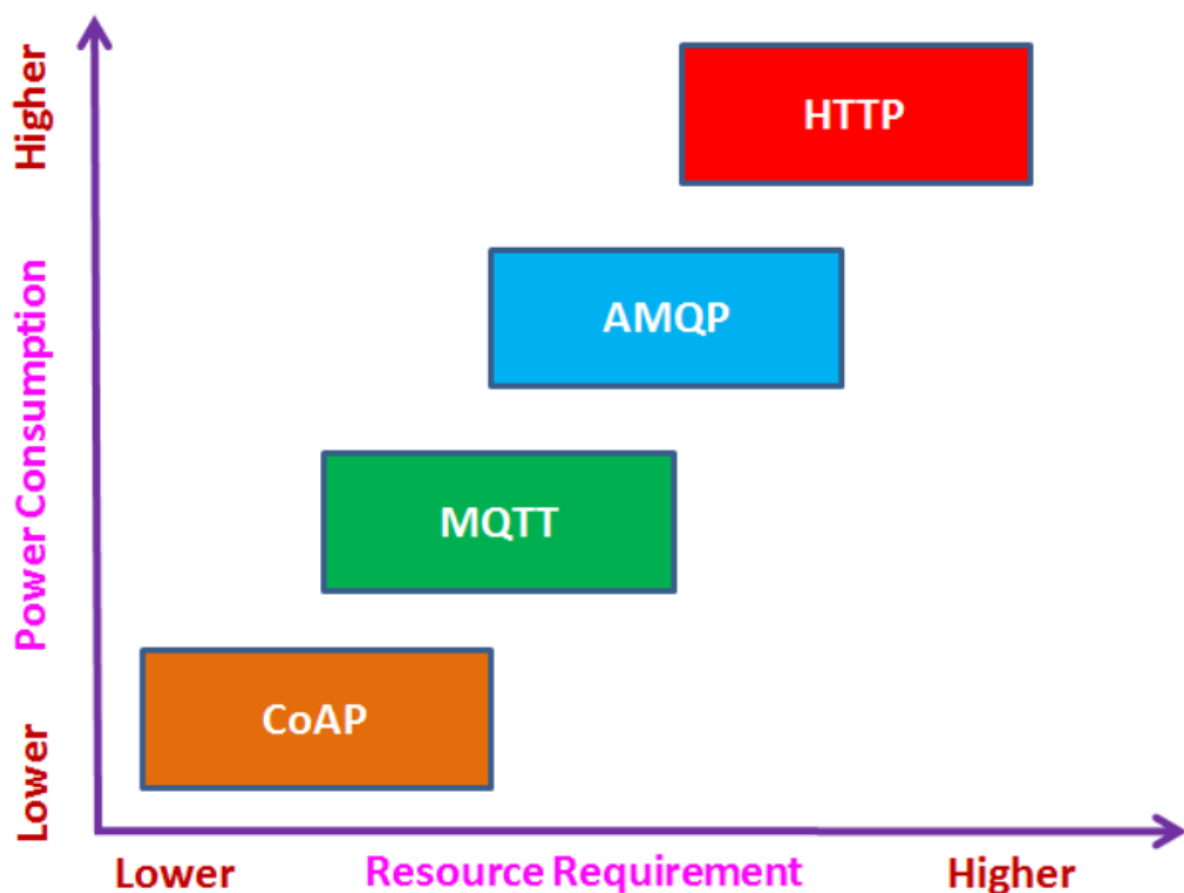
Fonte: [http://nodemcu.com/images/thumbnail/z1s\\_1.jpg\\_450x300.jpg](http://nodemcu.com/images/thumbnail/z1s_1.jpg_450x300.jpg).

Acessado em 16/10/2018

### 3. TRABALHOS CORRELATOS

O trabalho “Choice of Effective Messaging Protocols for IoT Systems: MQTT, CoAP, AMQP and HTTP” (NAIK, 2017) realiza um comparativo sobre entre os protocolos MQTT, *Constrained Application Protocol* (CoAP), *Advanced Message Queuing Protocol* (AMQP) e *HyperText Transfer Protocol* (HTTP), ressaltando pontos fortes e fracos de cada um deles, segundo com um comparativo prático e análise dos resultados. A figura 12 apresenta consumo de energia e requisitos de recursos dentre os critérios de análise para comparação.

Figura 12: Comparação entre protocolos.



Fonte: NAIK, 2017.

O presente trabalho se difere focando em apenas um protocolo citado pelo autor, o protocolo MQTT, ressaltando aspectos de segurança empregados para o protocolo.

O autor do trabalho “Comparison of security level and current consumption of security implementations for MQTT” (CARLSSON, ERIKSSON, 2018) aborda aspectos de segurança e consumo energético em dispositivos em Internet das Coisas (IoT) quando utilizado diferentes técnicas de segurança aplicadas ao protocolo MQTT, focando em alternativas ao uso de TLS, por afirmar que o mesmo demandaria grande carga de recursos computacionais para implantação nos dispositivos.

O presente trabalho é focado no estudo do TLS com certificado sobre o protocolo MQTT, uma vez que o dispositivo escolhido no trabalho possui recursos de processamento para realizar a tarefa proposta utilizando tal técnica.

O autor do artigo “Sistema De Monitoramento De Consumo De Água Utilizando O Protocolo De Comunicação MQTT” (MELO *et al*, 2018)” utiliza o protocolo MQTT sobre dispositivos IoT para enviar dados coletados a uma central para processamento posterior e elaboração de relatórios para análise, no contexto de monitoramento de utilização de água.

O presente trabalho também utiliza o protocolo MQTT, porém com técnicas de segurança implementadas, diferente do autor do artigo que não utiliza TLS ou qualquer outra forma de segurança implementada ao projeto.

O autor da dissertação “Uma análise comparativa dos protocolos SNMP, Zabbix e MQTT, no contexto de aplicações de Internet das Coisas” (MOTA, 2017) dá enfoque a aspectos de consumo de memória e energia aos dispositivos. No trabalho

é criado um dispositivo IoT para testes e executando os protocolos citados no mesmo, para obter as medições de consumo de energia e memória.

No presente trabalho, apesar de utilizar o dispositivo ESP8266 e o protocolo MQTT, apenas o MQTT é utilizado, ao invés de comparado a outros protocolos. Além disso, aspectos de segurança são levados em consideração na utilização dos protocolos.

O artigo “Weather Tracking System using MQTT and SQLite” (KODALI, GORANTLA, 2017) propõe um sistema de monitoramento do tempo e clima utilizando sensores de temperatura e umidade, que pelo protocolo MQTT registram as informações em um banco de dados para posterior apresentação. O artigo consiste no desenvolvimento de um framework para receber os dados do *broker* e inclusão no banco de dados.

O presente trabalho utiliza um ambiente semelhante, fazendo uso do protocolo MQTT, bem como um *broker*, mas com ênfase na segurança, enquanto o autor do artigo focou na implementação do receptor dos dados do lado do *broker*, não mencionando a segurança dos dados trafegando pela rede, ou criptografia aplicada ao protocolo.

O autor do artigo “Secure MQTT for Internet of Things (IoT)” (SINGH, 2015) propõe um esquema de segurança sobre o protocolo MQTT utilizando criptografia baseada em atributos sobre curvas elípticas. O autor defende a escolha na proposta citando a capacidade de enviar uma mensagem criptografada para diversos usuários simultaneamente, sem a necessidade de criptografar a mesma mensagem múltiplas vezes para então enviá-las individualmente, tornando viável para o ambiente onde foi testado, com múltiplos dispositivos IoT heterogêneos entre si,

conectados a um *broker*, sem a necessidade de múltiplas chaves públicas e privadas ou certificados para obter uma comunicação segura entre os dispositivos e o *broker*. Menções feitas ao protocolo TLS/SSL utilizando certificados e chaves de sessão é citada como trabalhosa do ponto de vista da complexidade de manutenção dos certificados e chaves necessária.

O presente trabalho contém uma quantidade menor de dispositivos IoT, tornando a abordagem viável utilizando o protocolo MQTT com TLS utilizando certificados para manter a organização do ambiente, e a confiabilidade do tráfego.

O autor do artigo “A Study on IOT Approach for Monitoring Water Quality Using MQTT Algorithm” (ABUBAKER et al, 2018) propõe o uso do protocolo MQTT para monitoramento da qualidade de água, poupando custos e trabalho manual em laboratório para determinar a qualidade da água. O modelo de obtenção de dados *Global System for Mobile Communications* (GSM) foi substituído pelo MQTT, devido a diversas limitações, como custos elevados, falhas de segurança e atrasos na transmissão dos dados.

Sendo assim, o autor opta por substituir a atual tecnologia pelo protocolo MQTT por ser escalável, eficiente e seguro, além de garantir a comunicação em ambos os sentidos.

O presente trabalho apesar de utilizar o protocolo MQTT na implementação da solução, utiliza além disso o protocolo TLS para obter mais segurança na comunicação entre sensores e servidor, contando também com um ambiente diferente de desenvolvimento, e um outro problema para solucionar.

O autor do artigo “The Day After Mirai: A Survey on MQTT Security Solutions After the Largest Cyber-attack Carried Out through an Army of IoT Devices”

(PERRONE et al, 2017) faz uma análise de soluções de segurança para IoT utilizando o protocolo MQTT, motivado pelo ataque do botnet mirai, que uma vez tendo acesso ao dispositivo, o transforma em uma máquina zumbi, podendo assim utilizá-la para posterior ataque em massa.

O artigo discorre sobre diferentes abordagens de segurança focadas no protocolo MQTT, incluindo a abordagem utilizando TLS, e com uso de certificado, dentre outras técnicas de segurança.

O presente trabalho utiliza ideias semelhantes, com o uso do protocolo MQTT com TLS e certificados digitais, acrescentando a prototipação de um ambiente e testes de transmissão de dados utilizando o protocolo.

A tabela 2 apresenta os trabalhos correlatos e seus respectivos tópicos, e ao final do mesmo, o presente trabalho contendo todos os tópicos.

Tabela 2: Trabalhos Correlatos.

Trabalho	IoT	MQTT	TLS
Choice of Effective Messaging Protocols for IoT Systems: MQTT, CoAP, AMQP and HTTP	X	X	
Comparison of security level and current consumption of security implementations for MQTT	X	X	
Sistema De Monitoramento De Consumo De Água Utilizando O Protocolo De Comunicação MQTT	X	X	
Uma análise comparativa dos protocolos SNMP, Zabbix e MQTT, no contexto de aplicações de Internet das Coisas	X	X	
Weather Tracking System using MQTT and SQLite	X	X	
Secure MQTT for Internet of Things (IoT)	X		X

A Study on IOT Approach for Monitoring Water Quality Using MQTT Algorithm	X	X	
The Day After Mirai: A Survey on MQTT Security Solutions After the Largest Cyber-attack Carried Out through an Army of IoT Devices	X		X
Presente Trabalho	X	X	X

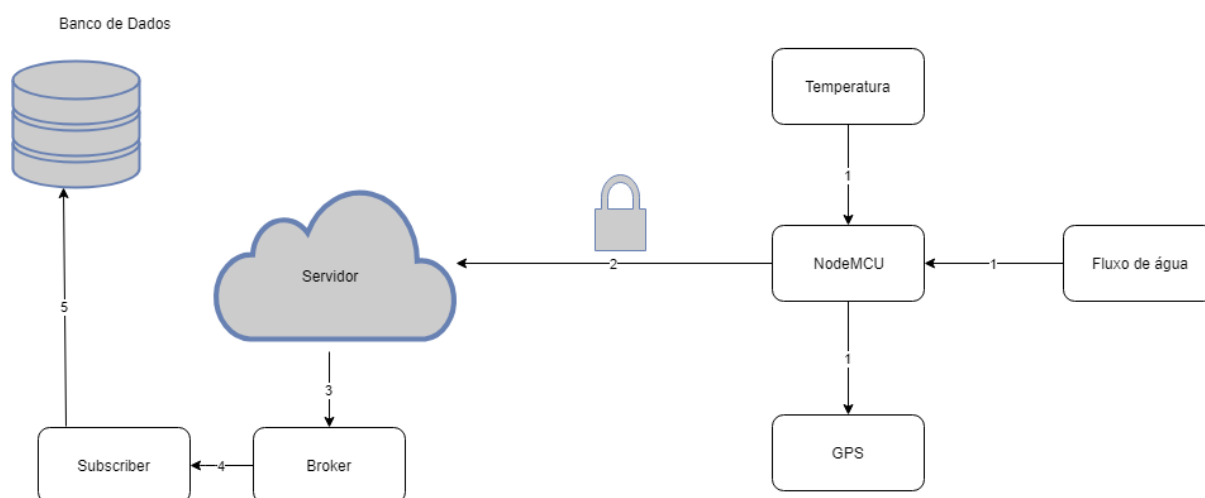
Fonte: Autor, 2018.

## 4. DESENVOLVIMENTO

Para a implementação do projeto proposto, diversas ferramentas são necessárias, como ambientes de desenvolvimento (IDE) para as linguagens Arduino, C++ e Python. O Arduino IDE é utilizado para a linguagem do Arduino, e Sublime para as demais linguagens citadas, além do terminal do sistema operacional para a execução de comandos específicos. As ferramentas citadas dão suporte a implementação do projeto proposto, operando parcialmente no *gateway*, e a outra parte em um ambiente na nuvem.

A figura 13 mostra o ambiente de desenvolvimento proposto, ilustrando as trocas de mensagens que acontecem ao longo do processo. Os passos 1 representam o envio dos dados aos sensores. O passo 2 representa o envio dos dados utilizando o protocolo MQTT com TLS do microcontrolador para o servidor na nuvem. O passo 3 é o redirecionamento dos dados para o *broker*. O passo 4 visa enviar os dados para um *subscriber* dentro do próprio servidor, para que no passo 5 o *subscriber* envie os dados coletados a um banco de dados.

Figura 13: Ambiente de desenvolvimento.



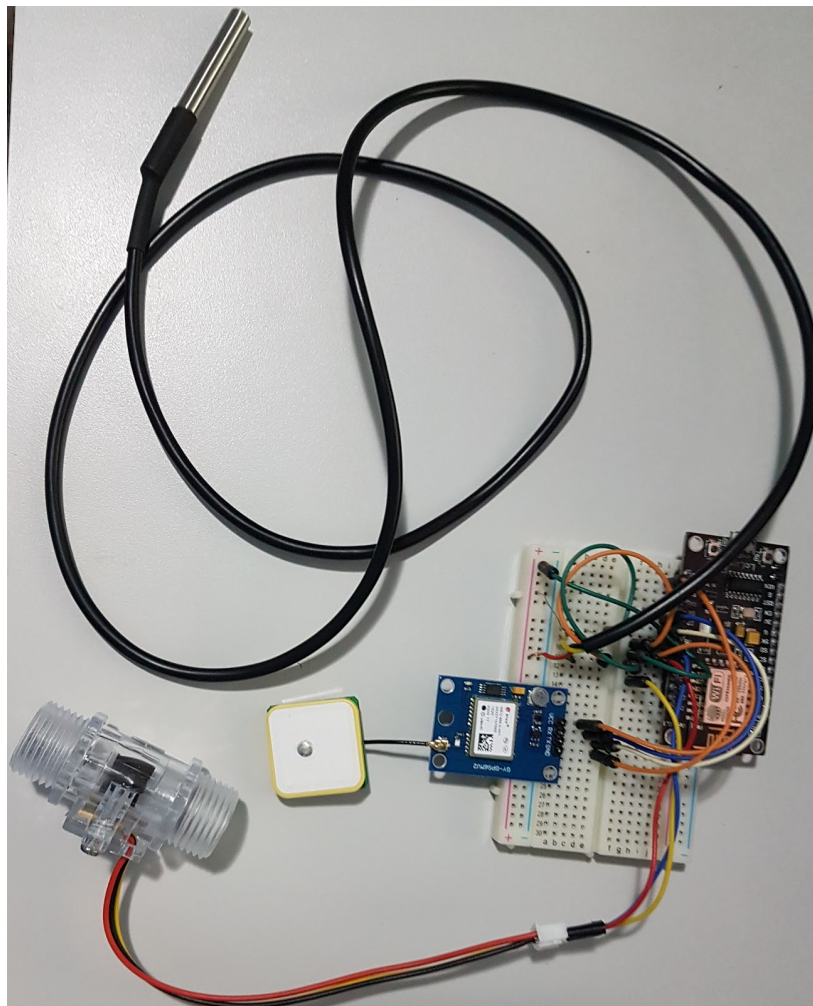
Fonte: Autor, 2018.



## 4.1 NODEMCU, SENSORES E GATEWAY

O NodeMCU se mostrou o microcontrolador mais adequado para suprir as necessidades no ambiente, pois uma vez que possui acesso nativo a WiFi, é capaz de agir como mediador entre os sensores e o servidor na nuvem, operando também no papel de *gateway*. No trabalho realizado, o NodeMCU atua como receptor dos dados coletados pelos sensores, medindo temperatura, fluxo de água e posicionamento no ambiente dos mesmos, e enviando ao servidor na nuvem os dados em um período de tempo definido.

Figura 14: Ambiente IoT.



Fonte: Autor, 2018.

A figura 14 ilustra o protótipo utilizado no trabalho, com o microcontrolador ligado aos sensores utilizados. O NodeMCU está ligado a uma *proto board*, plataforma pela qual são ligados ao microcontrolador, os sensores NEO-6M-0-001, YF-S201C e DS18B20, ligados através de fios e um resistor de 4.700 ohms com 5% de tolerância.

Cada sensor possui sua própria biblioteca, com funções específicas para obter os valores de medição dos mesmos, abstraindo detalhes de implementação referentes aos sensores. As bibliotecas DallasTemperature e OneWire são referentes ao sensor de temperatura, enquanto as bibliotecas TinyGPS e SoftwareSerial controlam o *Global Positioning System* (GPS). Outras bibliotecas são utilizadas para auxílio no envio de dados e segurança, com a biblioteca ESP8266WiFi realizando a conexão sem fios à Internet, PubSubClient realiza a transmissão de dados por meio do protocolo MQTT, FS faz o carregamento e a leitura do certificado no NodeMCU, e finalmente a biblioteca WiFiClientSecure faz a segurança, aplicando o protocolo TLS/SSL aos dados trafegando.

A figura 15 mostra um trecho do código implementado que é executado no microcontrolador NodeMCU. Esse trecho aparece logo após a inclusão das bibliotecas mencionadas anteriormente, e tem como finalidade a inicialização de variáveis que são utilizadas no decorrer do processo de obtenção e transferência dos dados dos sensores. O servidor na nuvem está especificado pela variável MQTT\_SERVER, e seu endereço público é 24.231.29.66.

Os métodos inicializados primeiramente são os de conexão com um ponto de acesso à Internet por meio de WiFi, com informações de nome do ponto de acesso

e senha. Em seguida são saídas físicas do microcontrolador são atribuídas à variáveis e objetos das classes das bibliotecas anteriormente citadas.

Figura 15: Código NodeMCU.

```
SoftwareSerial gpsSerial(12, 13);
TinyGPS gps;

#define MQTT_SERVER "35.231.29.66"
const char* mqtt_server = MQTT_SERVER;

const char* ssid = "*****";
const char* password = "*****";

#define ONE_WIRE_BUS D4 // PINO D4 DADOS - TEMP
const int buttonPin = D2; // PRINO D2 DADOS - Fluxo Agua

OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire); // Pass the oneWire reference to Dallas Temperature.

// Fingerprint of the broker CA
// openssl x509 -in mqttserver.crt -sha1 -noout -fingerprint
const char* fingerprint = "7C:CF:9A:37:7F:CA:57:D3:36:52:22:49:3D:3B:36:9E:FB:E8:97:11";

// Topic
char* topic = "SENSOR_01";

String clientName;

long lastReconnectAttempt = 0;
long lastMsg = 0;
int test_para = 900000; // ALTERAR O TEMPO!
unsigned long startMills;

// FLUXO AGUA
int contagem = 0; // variable to store the "rise ups" from the flowmeter pulses
int mls = 0;

String mac = "";

WiFiClientSecure wifiClient;
PubSubClient client(mqtt_server, 8883, wifiClient);

// METODOS DA CLASSE
void sendmqttMsg(char* topicToSend, String payload);
void verifytls();
void loadcerts();
boolean reconnect();
void wifi_connect();
String macToStr(const uint8_t* mac);
void init_Water_Flux();
void pin_ISR();
```

Fonte: Autor, 2018.

Após a inicialização das variáveis, o método *setup* é executado inicialmente por padrão no microcontrolador. Esse método chama outros métodos dentro de si, que por sua vez são responsáveis por estabelecer a conexão WiFi com o ponto de acesso sem fio para enviar os dados, e também por carregar o certificado previamente transferido ao NodeMCU e inicializar o sensor de fluxo de água.

Após o método *setup* ser executado por completo, o método *loop* é executado repetidamente enquanto o microcontrolador estiver ligado, repetindo as

instruções a cada ciclo do método. Este método primeiramente verifica se a conexão com o ponto de acesso está ativa, reconecta quando necessário, além de fazer a mesma verificação quando a conexão com o servidor tenha sido perdida. O método segue a sua execução lendo os dados dos sensores, concatenando para formar o *payload* que será enviado. No trabalho proposto, o *payload* é formado pelo endereço físico do microcontrolador, seguido da temperatura, fluxo de água e posicionamento por GPS. Os dados são enviados a cada 15 minutos, ou 90000 milissegundos, como inicializado em variável no código, chamando o método *sendmqttMsg*, passando o *payload* como parâmetro e tópico inicializado anteriormente.

A figura 16 mostra o método que envia o *payload* concatenado ao *broker* MQTT, por meio do comando *client.publish(topic\_send, payload, msg\_length)*, onde *topic\_send* é o parâmetro tópico, necessário para que o *broker* envie o mesmo aos *subscribers* com o mesmo tópico, seguido do *payload*, com os dados dos sensores, e um marcador com o tamanho da mensagem.

Figura 16: Método de envio dos dados ao Broker.

```
void sendmqttMsg(char* topic_send, String payload)
{
    if (client.connected()) {
        Serial.print("Sending payload: ");
        Serial.print(payload);

        unsigned int msg_length = payload.length();

        Serial.print(" length: ");
        Serial.println(msg_length);

        byte* p = (byte*)malloc(msg_length);
        memcpy(p, (char*) payload.c_str(), msg_length);

        if ( client.publish(topic_send, p, msg_length)) {
            Serial.println("Publish ok");
            free(p);
            //return 1;
        } else {
            Serial.println("Publish failed");
            free(p);
            //return 0;
        }
    }
}
```

Fonte: Autor, 2018.

## 4.2 BROKER

O *broker*, que recebe os dados do sensor, os envia ao subscriber de mesmo tópico em relação ao publisher. O *broker* escolhido foi o *Mosquitto*, por se tratar de um *software* de código aberto, de fácil implantação e configuração, viabilizando o uso de TLS com certificado, de acordo com as especificações do trabalho.

A figura 17 mostra a configuração feita no *broker* para viabilizar o uso de TLS com certificado. O comando *listener 8883* ordena ao *broker* que receba conexões pela porta 8883, que por convenção é a porta utilizada pelo protocolo MQTT para conexões com TLS. Abaixo dessa linha são referenciados os certificados e chaves pertencentes ao *broker*, utilizadas para validar o certificado dos clientes conectados. No final do arquivo consta a versão do TLS utilizada, explicitando qual versão do protocolo o *broker* deve utilizar.

Figura 17: Arquivo de configuração do broker.

```
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

pid_file /var/run/mosquitto.pid

persistence false
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

include_dir /etc/mosquitto/conf.d

listener 8883
cafile /etc/mosquitto/certs/ca.crt
certfile /etc/mosquitto/certs/mosquitto.crt
keyfile /etc/mosquitto/certs/mosquitto.key
require_certificate false
tls_version tlsv1.2
```

Fonte: Autor, 2018.

## 4.3 SUBSCRIBER

O cliente para onde o *broker* envia os dados foi escrito em *python*, se conectando ao *broker*, localizado no mesmo *host*, recebendo os dados e populando um banco de dados também no mesmo *host*, adicionando data e hora da obtenção dos dados para posterior referência. A figura 18 mostra parte do código do *subscriber*, inicialmente se conectando ao banco de dados e ao *broker* MQTT, utilizando o TLS com o certificado para garantir a recepção segura das mensagens. Em seguida, o código recebe o conteúdo do *broker* pela função *on\_message*, tratando o *payload* recebido e executando uma *query* na linguagem *Structured Query Language* (SQL) para inserir os dados ao banco.

Figura 18: Subscriber MQTT.

```
import paho.mqtt.client
import sys
import MySQLdb
import ssl

try:
    db = MySQLdb.connect("localhost", "*****", "*****", "IoT_MQTT")
except:
    print("Não se pode conectar ao servidor de banco de dados")
    print("TERMINANDO...")
    sys.exit()

cursor = db.cursor()

def on_connect(client, userdata, flags, rc):
    print('CONECTADO (%s)' % client._client_id)
    client.subscribe(topic='#', qos=2)

def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))
    lista = msg.payload.split("/")
    #
    sql = 'INSERT INTO IoT_MQTT.dados(sensor,mac_sensor,temp,fluxo_agua,gps,data_ins) VALUES (''
    CURRENT_TIMESTAMP);'
    try:
        cursor.execute(sql)
        db.commit()
    except:
        db.rollback()
        print("Falha ao gravar no Banco de Dados!")

client = paho.mqtt.client.Client(client_id='client-3', clean_session=False)
#client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

try:
    client.tls_set('/home/autenticacao/sslopen/mosquitto.crt', tls_version=ssl.PROTOCOL_TLSv1_2)
    client.connect(host='35.231.29.66', port=8883)
except:
    print("não é possível conectar ao MQTT Broker...")
    print("TERMINANDO...")
    db.close()
    sys.exit()

try:
    client.loop_forever()
except KeyboardInterrupt: #precionar Ctrl + C para sair
    print("FINALIZANDO...")
    db.close()
```

Fonte; Autor, 2018.

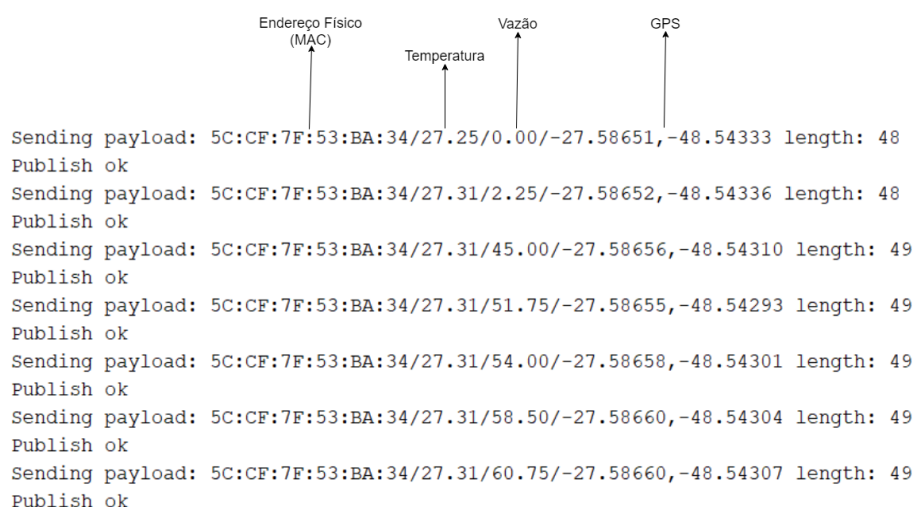
## 5. Análise de Resultados

Com o ambiente pronto, foram realizados alguns testes, cujos resultados geraram conteúdo para análise do ambiente. O teste foi realizado com o ambiente conforme a especificação, de forma com que funcionasse sem erros.

O teste tem como objetivo mostrar que o sistema como um todo produz o resultado esperado, tanto do lado do dispositivo com a conexão e envio dos dados de forma correta, quanto do lado do servidor, com a recepção e armazenamento.

O primeiro passo para o teste é a coleta de dados pelo dispositivo IoT, em um ambiente controlado, com acesso constante à Internet em um ambiente aberto. O dispositivo passou aproximadamente 48h coletando dados do ambiente e enviando ao servidor. A figura 19 mostra a saída textual do nodeMCU pela IDE ArduinoIDE, que é possível visualizar os dados que estão sendo enviados, juntamente com uma mensagem de retorno, afirmando que os dados foram enviados com sucesso. Os dados são concatenados por "/" na mensagem final, iniciando com o endereço físico da mensagem, seguido pelas medições de temperatura, fluxo e GPS, respectivamente.

Figura 19: Console NodeMCU.



```
Sending payload: 5C:CF:7F:53:BA:34/27.25/0.00/-27.58651,-48.54333 length: 48
Publish ok
Sending payload: 5C:CF:7F:53:BA:34/27.31/2.25/-27.58652,-48.54336 length: 48
Publish ok
Sending payload: 5C:CF:7F:53:BA:34/27.31/45.00/-27.58656,-48.54310 length: 49
Publish ok
Sending payload: 5C:CF:7F:53:BA:34/27.31/51.75/-27.58655,-48.54293 length: 49
Publish ok
Sending payload: 5C:CF:7F:53:BA:34/27.31/54.00/-27.58658,-48.54301 length: 49
Publish ok
Sending payload: 5C:CF:7F:53:BA:34/27.31/58.50/-27.58660,-48.54304 length: 49
Publish ok
Sending payload: 5C:CF:7F:53:BA:34/27.31/60.75/-27.58660,-48.54307 length: 49
Publish ok
```

Fonte: Autor, 2018.

Já no servidor na nuvem, o *script* que contém o *subscriber* recebe os mesmos dados enviados pelo dispositivo IoT, identificando também o sensor de onde os dados estão vindo. A figura 20 mostra a saída do *script*, que pode também ser executado sem saída, em segundo plano, mostrando primeiramente o estado de conexão, para então começar a receber os dados do dispositivo.

Figura 20: Saída Subscriber.

```
autenticacao@instance-1:~/mqtt_mysql$ python mqtt_mysql_local.py
CONECTADO (client-3)
SENSOR_01 5C:CF:7F:53:BA:34/28.19/0.00/-27.58665,-48.54287
SENSOR_01 5C:CF:7F:53:BA:34/28.56/0.00/0.00000,0.00000
SENSOR_01 5C:CF:7F:53:BA:34/28.56/0.00/-27.58668,-48.54287
SENSOR_01 5C:CF:7F:53:BA:34/28.62/0.00/-27.58670,-48.54286
SENSOR_01 5C:CF:7F:53:BA:34/28.62/0.00/-27.58671,-48.54283
SENSOR_01 5C:CF:7F:53:BA:34/28.62/0.00/-27.58670,-48.54281
SENSOR_01 5C:CF:7F:53:BA:34/28.62/0.00/-27.58669,-48.54281
SENSOR_01 5C:CF:7F:53:BA:34/28.62/0.00/-27.58672,-48.54281
SENSOR_01 5C:CF:7F:53:BA:34/28.62/0.00/-27.58675,-48.54281
SENSOR_01 5C:CF:7F:53:BA:34/28.62/0.00/-27.58679,-48.54282
SENSOR_01 5C:CF:7F:53:BA:34/28.62/0.00/-27.58680,-48.54282
SENSOR_01 5C:CF:7F:53:BA:34/28.62/0.00/-27.58681,-48.54284
SENSOR_01 5C:CF:7F:53:BA:34/28.62/0.00/-27.58679,-48.54280
```

Fonte: Autor, 2018.

Para fins de teste, um *subscriber* foi executado em um computador pessoal, utilizando o sistema operacional *Windows*, e utilizando a ferramenta de monitoramento e tráfego de rede *Wireshark*, é possível visualizar os dados que chegam ao computador por meio do protocolo TLS. A ferramenta *wireshark* faz uma análise em tempo real do tráfego de rede, no qual esse foi o meio utilizado para verificar que o *handshake* TLS é realizado com sucesso, provando que as mensagens trocadas entre as partes é de fato segura.

A figura 21 mostra as mensagens de cada passo do *handshake* com o servidor na nuvem, bem como mensagens de dados, contendo as medições feitas pelos sensores. Os pacote de número 3 a 5 mostram o *handshake* TCP, sendo esses pacotes SYN, ACK SYN e ACK, respectivamente.



Figura 21: Mensagens Wireshark.

No.	Source	Destination	Protocol	Info
3	192.168.0.197	35.231.29.66	TCP	58774 → 8883 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
4	35.231.29.66	192.168.0.197	TCP	8883 → 58774 [SYN, ACK] Seq=0 Ack=1 Win=28400 Len=0 MSS=1420 SACK_PERM=1 WS=128
5	192.168.0.197	35.231.29.66	TCP	58774 → 8883 [ACK] Seq=1 Ack=1 Win=66560 Len=0
6	192.168.0.197	35.231.29.66	TLSv1.2	Client Hello
7	35.231.29.66	192.168.0.197	TCP	8883 → 58774 [ACK] Seq=1 Ack=198 Win=29568 Len=0
8	35.231.29.66	192.168.0.197	TLSv1.2	Server Hello, Certificate, Server Key Exchange, Server Hello Done
9	192.168.0.197	35.231.29.66	TLSv1.2	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
10	35.231.29.66	192.168.0.197	TLSv1.2	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
11	192.168.0.197	35.231.29.66	TLSv1.2	Application Data
12	35.231.29.66	192.168.0.197	TLSv1.2	Application Data

Fonte: Autor, 2018.

Em seguida é feito o *handshake* TLS, mostrado na figura 20, nas mensagens de número 6, 8, 9 e 10. A mensagem 6 representa o *client Hello*, que mostra uma lista contendo os conjuntos criptográficos disponíveis pelo protocolo para realizar o *handshake*, conforme ilustrado na figura 21.

Na mensagem 8 acontece o *Server Hello*, juntamente com o envio do certificado do servidor para o cliente, bem como o início da troca de chaves. No experimento, o protocolo escolhido na mensagem, que aparece em destaque na figura 22, foi o seguinte: TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384 (0xc030).

Figura 22: Lista de conjuntos criptográficos.

```
Cipher Suites (28 suites)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
Cipher Suite: TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (0x009f)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xc0aa9)
Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xc0aa8)
Cipher Suite: TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xc0aa7)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
Cipher Suite: TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (0x009e)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (0xc024)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)
Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 (0x006b)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (0xc023)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027)
Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 (0x0067)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA256 (0x003d)
Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA256 (0x003c)
Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
```

Fonte: Autor, 2018.

As mensagens 8 e 9 definem a chave de sessão e estabelecimento de conexão TLS, com a primeira mensagem criptografada com o novo *ticket* utilizando o conjunto de cifras acordado entre as partes. A mensagem 12 é a primeira mensagem contendo os dados dos sensores vindas do servidor, listada no *wireshark* como sendo do protocolo TLSv1.2.

Além do teste de sucesso, foram feitos também testes de segurança em casos específicos de certificado e resumo de certificado incorretos. Para executar tais testes, foram feitas modificações no código, alterando primeiramente o valor do resumo, e posteriormente a carga de um certificado inválido para verificar se há conexão em tais casos.

Figura 23: Resumo inválido.

```
==> WIFI ---> Connecting to NET 52
. 0. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14
==> WiFi connected -----> IP address: 192.168.0.8
Success to open cert file
cert not loaded
connecting to 35.231.29.66
certificate doesn't match
```

Fonte: Autor, 2018.

O código do sensor possui uma linha que especifica um *fingerprint*, um resumo do certificado do servidor, e seu conteúdo foi trocado para o teste, que responde com a mensagem ilustrada na figura 23.

Para o último teste realizado, o caminho para o certificado foi trocado, nesse caso, o cliente também não deverá se conectar ao servidor. A figura 24 mostra a resposta do dispositivo IoT após tal modificação, sendo impossível carregar o arquivo de certificado.

Figura 24: Certificado inválido.

```
==> WIFI ---> Connecting to NET 52  
. 0. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14.  
==> WiFi connected -----> IP address: 192.168.0.8  
Failed to open cert file
```

Fonte: Autor, 2018.

## 6. CONCLUSÃO E TRABALHOS FUTUROS

O trabalho aqui discutido é frequentemente atualizado, novas tecnologias e formas de ataque surgem de tempos em tempos. Em relação ao que foi proposto no trabalho, com a tecnologia atual, mostrou um resultado satisfatório no teste proposto, provando sua eficácia.

Em relação ao ambiente proposto, os sensores representaram o ambiente de forma objetiva e precisa, dando credibilidade ao objetivo proposto para solucionar o problema.

No que diz respeito ao objetivo de enviar os dados de forma segura, pode-se verificar que o protocolo TLS garante a segurança dos dados trafegando pela rede, provendo integridade e confidencialidade aos dados entre os pontos.

Sobre o monitoramento em si, o tempo em que o ambiente foi monitorado garantiu ao projeto consistência e providência de dados reais para a análise dos resultados, sendo satisfatório para o que se propõe.

Os resultados do trabalho mostram-se conclusivos, tanto do ponto de vista do monitoramento e visualização dos dados, quanto do ponto de vista da segurança implementada.

Dentre as possibilidades de trabalho que podem ser realizadas a partir deste, tanto em Internet das Coisas quanto em segurança da informação, é possível realizar estudos de escalabilidade do ambiente, adicionando diversos microcontroladores a pulverizadores diferentes, analisando mais de um grupo de sensores de uma só vez. Também é possível abranger outros locais para estudo, testando a eficácia do sistema em diferentes regiões. Outra possível extensão do ambiente seria a adição de controladores de fluxo no dispositivo, como por exemplo,

a ação de aumento ou diminuição na vazão de fungicida remotamente, utilizando sensores específicos no ambiente IoT.

## REFERENCIAS

ABUBAKER, Alfiya et al. **A Study on IOT Approach for Monitoring Water Quality Using MQTT Algorithm.** 2018.

BONOMI, Flavio et al. Fog computing and its role in the internet of things. In: **Proceedings of the first edition of the MCC workshop on Mobile cloud computing.** ACM, 2012. p. 13-16.

CARLSSON, Fredrik; ERIKSSON, Klas-Göran. **Comparison of security level and current consumption of security implementations for MQTT.** 2018.

NAIK, Nitin. **Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP.** In: Systems Engineering Symposium (ISSE), 2017 IEEE International. IEEE, 2017. p. 1-7.

CASTELUCCI, Daniella. 2011. **PROTOCOLOS DE COMUNICAÇÃO EM REDES DE COMPUTADORES.** Disponível em:

<https://daniellacastelucci.wordpress.com/2011/04/08/protocolos-de-comunicacao-em-redes-de-computadores>. Acessado em: 14/11/2018.

DO ESPÍRITO SANTO, Adrielle Fernanda Silva. **Segurança da informação.** Instituto Cuiabano de Educação (ICE), 2011.

FILHO, Socrates. **Segurança da Informação: Autenticação.** 2009. Disponível em: <http://waltercunha.com/blog/2009/08/19/seguranca-da-informacao-autenticacao>. Acessado em: 14/11/2018.

GUGELMIN, Felipe. 2014. **Internet: qual a diferença entre os protocolos UDP e TCP?.** Disponível em:

<https://www.tecmundo.com.br/internet/57947-internet-diferenca-entre-protocolos-udp-tcp.htm>. Acessado em: 14/11/2018.

HALLER, Stephan; KARNOUSKOS, Stamatis; SCHROTH, Christoph. **The Internet of things in an enterprise context**. In: **Future Internet Symposium**. Editora

Springer Berlin Heidelberg, 2008. p. 14-28.

KODALI, Ravi Kishore; GORANTLA, Venkata Sundeep Kumar. **Weather tracking system using MQTT and SQLite**. In: 2017 3rd International Conference on Applied

and Theoretical Computing and Communication Technology (iCATccT). IEEE, 2017.

p. 205-208.

MELO, Rodrigo Cassiano da Silva et al. **Sistema de monitoramento de consumo de água utilizando o protocolo de comunicação MQTT**. 2018.

MOTA, Levi da Costa. **Uma análise comparativa dos protocolos SNMP, Zabbix e MQTT, no contexto de aplicações de Internet das Coisas**. 2017.

OASIS, **MQTT Version 3.1.1**. 2014. Disponível em:

<http://docs.oasisopen.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf> acessado em:

10/11/2016.

PERRONE, Giovanni et al. The Day After Mirai: A Survey on MQTT Security

Solutions After the Largest Cyber-attack Carried Out through an Army of IoT

Devices. In: 2nd International Conference on Internet of Things, Big Data and

Security. 2017. p. 246-253.

RIDLEY, Daisy. 2017. **4 Ways in Which IoT is Revolutionising Agriculture**.

Disponível em: <https://datafloq.com/read/4-ways-iot-revolutionising-agriculture/4121>.

Acessado em 14/11/2018.

SINGH, Meena et al. **Secure mqtt for Internet of things (iot)**. In: Communication systems and network technologies (CSNT), 2015 fifth international conference on.

IEEE, 2015. p. 746-751.

STALLINGS, William. **Criptografia e segurança de redes: princípios e práticas.**

4. ed. São Paulo: Pearson Prentice Hall, 2008. 492 p. ISBN 9788576051190.

UCKELMANN, Dieter; HARRISON, Mark; MICHAHELLES, Florian. **An architectural approach towards the future Internet of things.** In: Architecting the Internet of things. Editora Springer Berlin Heidelberg, 2011. p. 1-24.

STALLINGS, William; BROWN, Lawrie. **Segurança de Computadores: Princípios e Práticas.** 2. ed. São Paulo: Elsevier, 2013. ISBN 9788535264494.

VON SOLMS, Rossouw; VAN NIEKERK, Johan. **From information security to cyber security.** computers & security, v. 38, p. 97-102, 2013.



## APÊNDICE A - CÓDIGO FONTE

### NodeMCU

```
#include <FS.h> // BIBLIOTECA PARA FUNCIONAR A SPIFFS
#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>
#include <PubSubClient.h>
#include <DallasTemperature.h>
#include <OneWire.h>
#include <SoftwareSerial.h>
#include "TinyGPS++.h"

TinyGPSPlus gps;                                // Create an Instance of the TinyGPS++
object called gps
SoftwareSerial gpsSerial(12, 13);

#define MQTT_SERVER "35.231.29.66"
const char* mqtt_server = MQTT_SERVER;

const char* ssid = "NET 52";
const char* password = "!123456#";

//const char* ssid = "LRG";
//const char* password = "$UFSC$LRG";

#define ONE_WIRE_BUS D4 // PINO D4 DADOS - TEMP
const int buttonPin = D2; // PRINO D2 DADOS - Fluxo Agua

OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);           // Pass the oneWire reference to
Dallas Temperature.

// Fingerprint of the broker CA
// openssl x509 -in mqttserver.crt -sha1 -noout -fingerprint
const char* fingerprint =
"7C:CF:9A:37:7F:CA:57:D3:36:52:22:49:3D:3B:36:9E:FB:E8:97:11";

// Topic
char* topic = "SENSOR_01";

String clientName;
```

```

long lastReconnectAttempt = 0;
long lastMsg = 0;
int test_para = 900000; // ALTERAR O TEMPO!
unsigned long startMills;

// FLUXO AGUA
int contagem = 0; // variable to store the "rise ups" from the flowmeter pulses
int mls = 0;

String mac = "";

WiFiClientSecure wifiClient;
PubSubClient client(mqtt_server, 8883, wifiClient);

// METODOS DA CLASSE
void sendmqttMsg(char* topicToSend, String payload);
void verifytls();
void loadcerts();
boolean reconnect();
void wifi_connect();
String macToStr(const uint8_t* mac);
void init_Water_Flux();
void pin_ISR();
String printFloat(double f, int digits = 2);

void setup(){
  Serial.begin(9600);
  sensors.begin();
  startMills = millis();
  delay(1500);
  gpsSerial.begin(9600); // Set Software Serial Comm Speed to
9600

  wifi_connect();
  delay(500);

  loadcerts();
  delay(200);
  verifytls();

  mac = WiFi.macAddress();

```

```

init_Water_Flux();
gps.charsProcessed();
}

void loop(){
  if (WiFi.status() == WL_CONNECTED) {
    if (!client.connected()) {
      long now = millis();
      if (now - lastReconnectAttempt > 2000) {
        lastReconnectAttempt = now;
        if (reconnect()) {
          lastReconnectAttempt = 0;
        }
      }
    } else {
      bool newdata = false;
      unsigned long start = millis();
      // Every 5 seconds we print an update
      while (millis() - start < 5000)
      {
        if (gpsSerial.available())

        {
          char c = gpsSerial.read();
          //Serial.print(c); // uncomment to see raw GPS data
          if (gps.encode(c))
          {
            newdata = true;
            break; // uncomment to print new data immediately!
          }
        }
      }
    }

    if (newdata)
    {
      float flat, flon;
      //gps.f_get_position(&flat, &flon);
      long now = millis();
      if (now - lastMsg > test_para) {
        sensors.requestTemperatures();
        lastMsg = now;
      }
    }
  }
}

```

```

    String payload = mac;
    payload += "/";
    payload += (sensors.getTempCByIndex(0));
    payload += "/";
    payload += (contagem*2.25);
    contagem = 0;
    payload += "/";
    payload += printFloat(gps.location.lat(), 5);
    payload += ",";
    payload += printFloat(gps.location.lng(), 5);
    sendmqttMsg(topic, payload);
  }
}
client.loop();
}
} else {
  wifi_connect();
}
}

void init_Water_Flux(){
  pinMode(buttonPin, INPUT);
  attachInterrupt(digitalPinToInterrupt(buttonPin), pin_ISR, RISING);
}

void pin_ISR(){
  contagem++;
}

void verifytls() {
  // Use WiFiClientSecure class to create TLS connection
  Serial.print("connecting to ");
  Serial.println(mqtt_server);
  if (!wifiClient.connect(mqtt_server, 8883)) {
    Serial.println("connection failed");
    return;
  }

  if (wifiClient.verify(fingerprint, mqtt_server)) {
    Serial.println("certificate matches");
  } else {
    Serial.println("certificate doesn't match");
  }
}

```

```

    }
}

// Load Certificates
void loadcerts() {

    if (!SPIFFS.begin()) {
        Serial.println("Failed to mount file system");
        return;
    }

    // Load client certificate file from SPIFFS
    File cert = SPIFFS.open("/mosquittoF.crt", "r"); //replace esp.der with your uploaded
file name
    if (!cert) {
        Serial.println("Failed to open cert file");
    }
    else
        Serial.println("Success to open cert file");

    delay(1000);

    // Set client certificate
    if (wifiClient.loadCertificate(cert))
        Serial.println("cert loaded");
    else
        Serial.println("cert not loaded");
}

boolean reconnect(){
    if (!client.connected()) {
        if (client.connect((char*) clientName.c_str())) {
            Serial.println("==> mqtt connected");
        } else {
            Serial.print("---> mqtt failed, rc=");
            Serial.println(client.state());
        }
    }
}
return client.connected();
}

void wifi_connect()

```

```

{
  if (WiFi.status() != WL_CONNECTED) {
    // WIFI
    Serial.println();
    Serial.print("==> WIFI ---> Connecting to ");
    Serial.println(ssid);

    delay(10);
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);

    int Attempt = 0;
    while (WiFi.status() != WL_CONNECTED) {
      Serial.print(". ");
      Serial.print(Attempt);

      delay(100);
      Attempt++;
      if (Attempt == 150){
        Serial.println();
        Serial.println("-----> Could not connect to WIFI");

        ESP.restart();
        delay(200);
      }
    }
    Serial.println();
    Serial.print("==> WiFi connected");
    Serial.print(" -----> IP address: ");
    Serial.println(WiFi.localIP());
  }
}

void sendmqttMsg(char* topicToSend, String payload)
{
  if (client.connected()) {
    Serial.print("Sending payload: ");
    Serial.print(payload);

    unsigned int msg_length = payload.length();
  }
}

```

```

Serial.print(" length: ");
Serial.println(msg_length);

byte* p = (byte*)malloc(msg_length);
memcpy(p, (char*) payload.c_str(), msg_length);

if ( client.publish(topicToSend, p, msg_length)) {
    Serial.println("Publish ok");
    free(p);
    //return 1;
} else {
    Serial.println("Publish failed");
    free(p);
    //return 0;
}
}
}

String printFloat(double number, int digits)
{
    String str;
    // Handle negative numbers
    if (number < 0.0)
    {
        str += "-";
        number = -number;
    }

    // Round correctly so that print(1.999, 2) prints as "2.00"
    double rounding = 0.5;
    for (uint8_t i=0; i<digits; ++i)
        rounding /= 10.0;

    number += rounding;

    // Extract the integer part of the number and print it
    unsigned long int_part = (unsigned long)number;
    double remainder = number - (double)int_part;
    String aux = String(int_part, DEC);
    str += aux;

```

```

// Print the decimal point, but only if there are digits beyond
if (digits > 0)
    str += ".";

// Extract digits from the remainder one at a time
while (digits-- > 0)
{
    remainder *= 10.0;
    int toPrint = int(remainder);
    str += String(toPrint);
    remainder -= toPrint;
}
return str;
}

```

```

String macToStr(const uint8_t* mac){
    String result;
    for (int i = 0; i < 6; ++i) {
        result += String(mac[i], 16);
        if (i < 5)
            result += ':';
    }
    return result;
}

```

```

String mac2String(byte ar[]){
    String s;
    for (byte i = 0; i < 6; ++i)
    {
        char buf[3];
        sprintf(buf, "%02X", ar[i]);
        s += buf;
        if (i < 5) s += ':';
    }
    return s;
}

```

### Broker

```

# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

```



```

pid_file /var/run/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

include_dir /etc/mosquitto/conf.d

listener 8883
cafile /etc/mosquitto/certs/ca.crt
certfile /etc/mosquitto/certs/mosquitto.crt
keyfile /etc/mosquitto/certs/mosquitto.key
require_certificate false
tls_version tlsv1.2

```

### **Subscriber**

```

#!/usr/bin/env python 1
# -*- coding: utf-8 -*-

import paho.mqtt.client
import sys
import MySQLdb
import ssl

try:
    db = MySQLdb.connect("localhost","loffi","lleandro","IoT_MQTT")
except:
    print("Não se pode conectar ao servidor de banco de dados")
    print("TERMINANDO...")
    sys.exit()

cursor = db.cursor()

def on_connect(client, userdata, flags, rc):
    print('CONECTADO (%s)' % client._client_id)
    client.subscribe(topic='#', qos=2)

def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))
    lista = msg.payload.split("/")

```

```

# SENSOR
MAC_SENSOR      TEMP      FLUXO      TEMPO
sql              =      'INSERT      INTO
IoT_MQTT.dados(sensor,mac_sensor,temp,fluxo_agua,gps,data_ins) VALUES ("+
str(msg.topic)+"", "+lista[0]"+",      '+lista[1]'+',      '+lista[2]'+', "+str(lista[3])+"',
CURRENT_TIMESTAMP);'
try:
    cursor.execute(sql)
    db.commit()
except:
    db.rollback()
    print("Falha ao gravar no Banco de Dados!")

client = paho.mqtt.client.Client(client_id='client-3', clean_session=False)
#client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

try:
    client.tls_set('/home/autenticacao/sslopen/mosquitto.crt',
tls_version=ssl.PROTOCOL_TLSv1_2)
    client.connect(host='35.231.29.66', port=8883)
except:
    print("não é possível conectar ao MQTT Broker...")
    print("TERMINANDO...")
    db.close()
    sys.exit()

try:
    client.loop_forever()
except KeyboardInterrupt: #precionar Ctrl + C para sair
    print("FINALIZANDO...")
    db.close()

```

## APÊNDICE B - ARTIGO SOBRE O TRABALHO

### Ambiente de comunicação segura de Internet das Coisas com a utilização do MQTT e TLS

Eduardo de Meireles Koneski<sup>1</sup>

<sup>1</sup>Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC) Caixa Postal 476 – 88.040-900 – Florianópolis – SC – Brazil

emeireleskoneski@gmail.com

**Abstract.** *This paper focuses on some of the key aspects of security, integrity, confidentiality and authentication. These principles are applied to the Internet of Things context, which is a paradigm that aims to use the Internet to connect devices to the world wide computer network, characterized by the use of an intermediary, such as a microcontroller. The proposal is divided into two main stages. The first one focuses on receiving sensor data and sending it to a server in the cloud. The second step is to receive the data from the first step and store it. For security requirements, MQTT protocols with TLS were used together to ensure integrity and confidentiality, and digital certificates ensure authentication. The present work carried out validation tests for both the correct transfer of the data, and of the relevant security aspects related to it.*

**Resumo.** *O presente trabalho foca em alguns dos principais aspectos de segurança, integridade, confidencialidade e autenticação. Estes princípios são aplicados ao contexto de Internet das Coisas, que é um paradigma que visa o uso da Internet para interligar dispositivos a rede mundial de computadores, caracterizado pelo uso de um intermediário, como um microcontrolador. A proposta é dividida em duas etapas principais. A primeira foca em receber dados de sensores e enviar a um servidor na nuvem. A segunda etapa consiste em receber os dados da primeira etapa e armazenar. Para atingir requisitos de segurança, os protocolos MQTT com TLS foram utilizados em conjunto para garantir integridade e confidencialidade, e certificados digitais asseguram a autenticação. O presente trabalho realizou testes de validação tanto da transferência correta dos dados, como de aspectos de segurança relevantes referentes ao mesmo.*

#### 1. Introdução

A utilização de ferramentas para auxílio na agricultura não é nova, e avanços tem sido feito a esse respeito com o uso da tecnologia.

A tecnologia tem auxiliado em diversos setores produtivos, em organizações de pequeno a grande porte. Nesse contexto, o uso de tecnologia no setor agrícola pode representar benefícios tanto do ponto de vista comercial, quanto produtivo, pois age em diversos setores da indústria.

Por uma perspectiva tecnológica, os dados que podem ser coletados em uma área agrícola são enviados a um servidor da Internet, no qual, outros usuários podem ter acesso e alterar valores de medição coletados pelos sensores por meio de interceptações dos dados.

Stallings (2008) cita duas formas de ameaças a sistemas - podendo ser de informação ou serviços - o autor segue explicando que ambos são programas escritos com a finalidade de explorar vulnerabilidades sobre os respectivos domínios.

Daisy Ridley (2017) insere IoT no contexto da agricultura e pecuária, projetando crescimento significativo na área de agricultura inteligente, devido a adesão da IoT no ramo, impactando diretamente na minimização de custos e melhores resultados, como menor consumo de água e quantidade de perdas.

Dentre os variados cenários com o uso de dispositivos IoT, podem ser tanto em uma smart city como também em ambientes agrícolas. O principal problema apresentado em ambientes agrícolas segundo especialistas é como é possível monitorar áreas que são pulverizadas, mas mesmo assim surgem problemas com doenças e insetos de erva daninhas.

Problemas neste aspecto podem acarretar grandes perdas ao agricultor por não ter um dispositivo que é preciso e informe dados corretos.

Segurança da informação é um conjunto de ferramentas, políticas, conceitos, ações e práticas que podem ser utilizadas em conjunto com a finalidade de proteger um ambiente, seja ele organizacional ou pessoal, podendo agir sob dispositivos, aplicações, serviços, entre outros. [Von Solms and Van Niekerk, 2013].

Para solucionar o problema de insegurança no tráfego de dados na comunicação entre os dispositivos coletores de informações de vazão de fungicida e a *Cloud*, é proposto a solução de utilização do MQTT juntamente com o TLS/SSL, que usa o certificado digital para fazer a comunicação entre as partes.

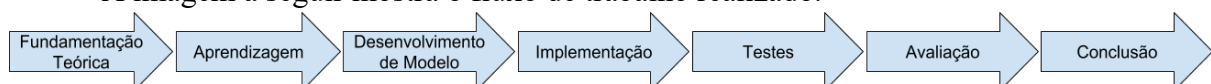
O protocolo da camada de aplicação *Message Queuing Telemetry Transport* (MQTT), foi criado na década de 90, com a premissa de ser simples, leve e de fácil implementação (OASIS, 2014).

Neste trabalho, é apresentado como a tecnologia pode ser utilizada para o fim de auxiliar no monitoramento de sensores, podendo ser útil nas etapas supracitada, e mantendo as informações protegidas.

## 2. Metodologia

Tendo problemas e objetivos definidos, foi adotada uma estratégia para obter o resultado esperado, conforme a Figura 1, que consiste nos seguintes passos: Revisão da literatura, aprendizagem dos temas que estão vinculados a uma rede IoT, desenvolvimento de um modelo de monitoramento em ambientes agrícola, projetar um ambiente com sensores para enviar dados utilizando o protocolo MQTT com TLS/SSL, testar o protótipo em ambiente real, avaliar o modelo proposto, e escrever os resultados do trabalho.

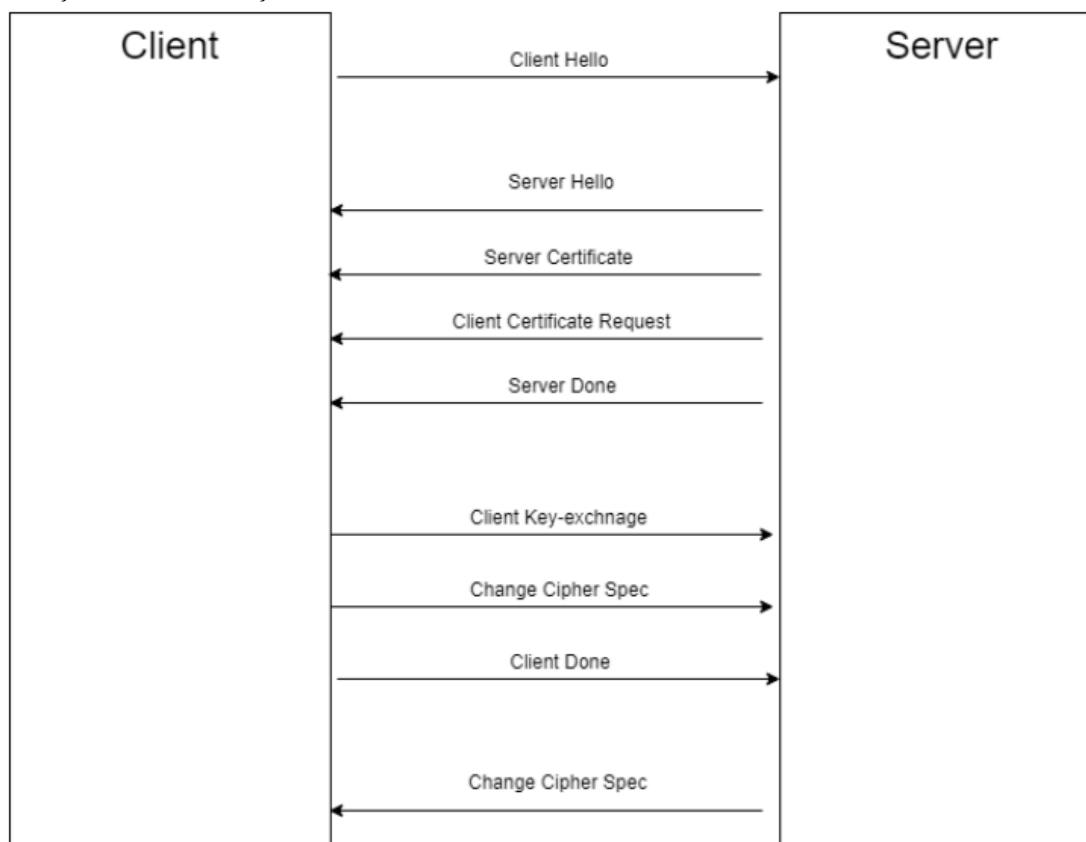
A imagem a seguir mostra o fluxo de trabalho realizado.



**Figura 1. Fluxo de metodologia**

Os trabalhos anteriores foram os principais pontos de partidas para a forma de obter informação e referencial teórico de autores para o desenvolvimento do trabalho. Os principais temas de pesquisa por referências foram MQTT, NodeMCU, TLS/SSL e segurança, abrangendo também combinações das palavras chave citadas.

A figura abaixo mostra os passos que caracterizam a segurança empregada ao protocolo TLS/SSL. O protocolo executa uma troca seguida de uma escolha de algoritmos criptográficos, seguido pela troca de chaves e verificação de certificados, atribuindo segurança na comunicação.



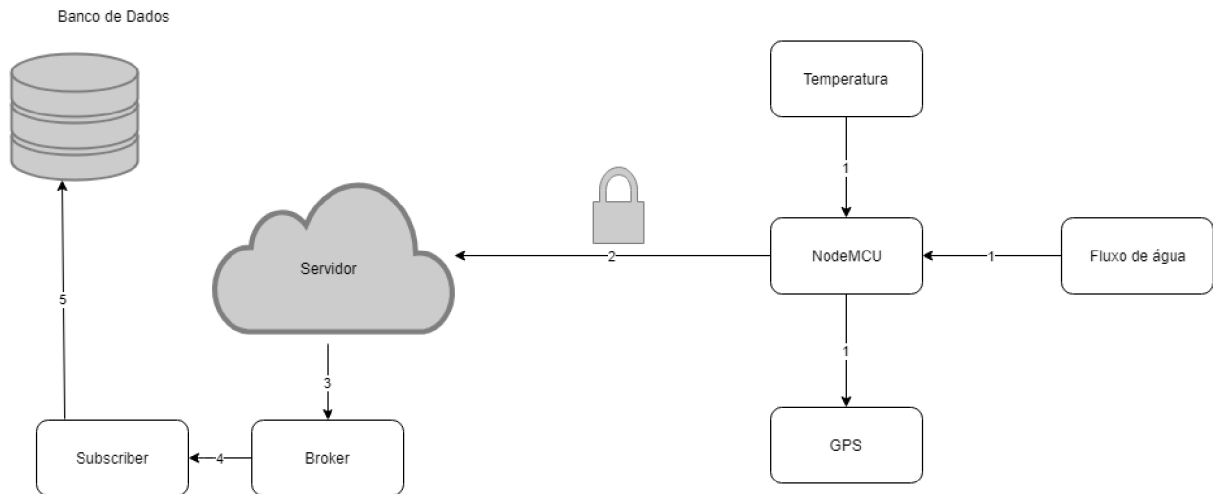
**Figura 2. Handshake TLS/SSL**

Para a realização do trabalho, foi necessário o aprendizado dos temas descritos no referencial teórico. Teve-se como principais temas de pesquisa as redes IoT, Fog Computing e sensores. Ainda foram utilizados outros temas específicos para a montagem do ambiente e implementação do código nos dispositivos IoT. Ou seja, foi aprendido a linguagem do Arduino, C++, Python e SQL para o desenvolvimento posterior de um ambiente real.

Em relação ao desenvolvimento do modelo de monitoramento em ambiente agrícola, foi utilizado o auxílio de um especialista na área de agronomia para melhor adequar em um ambiente real o modelo. O modelo partiu do princípio do interesse de monitorar áreas agrícolas que são pulverizadas.

Sensores sobre a plataforma de desenvolvimento NodeMCU foram utilizados, os dados obtidos são então enviados para um servidor na nuvem utilizando o protocolo MQTT na camada de transporte, e para a segurança foi utilizado o protocolo TLS/SSL com certificados.

Na figura abaixo é possível visualizar os elementos presentes no ambiente, numerados em passos para facilitar o entendimento do mesmo. O passo 1 representa a medição dos dados dos sensores. O passo 2 representa o envio dos dados do NodeMCU para o servidor na nuvem. O passo 3 se refere ao recebimento dos dados pelo *broker*. O passo 4 visa enviar os dados do *broker* ao *subscriber*. Finalmente no passo 5 os dados são enviados do *subscriber* ao banco de dados.



**Figura 3. Ambiente IoT**

Com o protótipo finalizado, testes foram feitos sobre ele, em um ambiente simulado. Os testes contemplam as medições dos sensores, o transporte dos dados ao gateway, e o protocolo de segurança implantado.

Com os dados coletados, a avaliação do modelo é feita verificando se os dados enviados são os mesmos recebidos.

Por fim é escrito os resultados adquiridos com o modelo proposto, levando em consideração a metodologia aqui descrita.

### 3. Análise de Resultados

Com o ambiente pronto, foram realizados alguns testes, cujos resultados geraram conteúdo para análise do ambiente. O teste foi realizado com o ambiente conforme a especificação, de forma com que funcionasse sem erros.

O teste tem como objetivo mostrar que o sistema como um todo produz o resultado esperado, tanto do lado do dispositivo com a conexão e envio dos dados de forma correta, quanto do lado do servidor, com a recepção e armazenamento.

O primeiro passo para o teste é a coleta de dados pelo dispositivo IoT, em um ambiente controlado, com acesso constante à Internet em um ambiente aberto. O dispositivo passou aproximadamente 48h coletando dados do ambiente e enviando ao servidor.

A próxima figura mostra os dados sendo enviados para o servidor na nuvem, concatenados por “/”. A mensagem enviada começa pelo endereço físico (MAC) do microcontrolador, seguido pelas medições de temperatura, fluxo e posicionamento (GPS), respectivamente.

```

Sending payload: 5C:CF:7F:53:BA:34/27.25/0.00/-27.58651,-48.54333 length: 48
Publish ok
Sending payload: 5C:CF:7F:53:BA:34/27.31/2.25/-27.58652,-48.54336 length: 48
Publish ok
Sending payload: 5C:CF:7F:53:BA:34/27.31/45.00/-27.58656,-48.54310 length: 49
Publish ok
Sending payload: 5C:CF:7F:53:BA:34/27.31/51.75/-27.58655,-48.54293 length: 49
Publish ok
Sending payload: 5C:CF:7F:53:BA:34/27.31/54.00/-27.58658,-48.54301 length: 49
Publish ok
Sending payload: 5C:CF:7F:53:BA:34/27.31/58.50/-27.58660,-48.54304 length: 49
Publish ok
Sending payload: 5C:CF:7F:53:BA:34/27.31/60.75/-27.58660,-48.54307 length: 49
Publish ok

```

**Figura 4. Saída do microcontrolador**

Já no servidor na nuvem, o script que contém o subscriber recebe os mesmos dados enviados pelo dispositivo IoT, identificando também o sensor de onde os dados estão vindo.

Para fins de teste, um subscriber foi executado em um computador pessoal, utilizando o sistema operacional *Windows*, e utilizando a ferramenta de monitoramento e tráfego de rede *Wireshark*, é possível visualizar os dados que chegam ao computador por meio do protocolo TLS. A ferramenta wireshark faz uma análise em tempo real do tráfego de rede, no qual esse foi o meio utilizado para verificar que o *handshake* TLS é realizado com sucesso, provando que as mensagens trocadas entre as partes é de fato segura.

A figura a seguir mostra a lista de conjuntos de cifras aceitas pelo *subscriber*, sendo a opção em destaque a escolhida para a comunicação segura.

```

Cipher Suites (28 suites)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
Cipher Suite: TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (0x009f)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xc0a9)
Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xc0aa8)
Cipher Suite: TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xc0aa)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
Cipher Suite: TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (0x009e)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (0xc024)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)
Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 (0x006b)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (0xc023)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027)
Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 (0x0067)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA256 (0x003d)
Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA256 (0x003c)
Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)

```

**Figura 5. Conjuntos de cifras**

Além do teste de sucesso, foram feitos também testes de segurança em casos específicos de certificado e resumo de certificado incorretos. Para executar tais testes, foram feitas modificações no código, alterando primeiramente o valor do resumo, e posteriormente

a carga de um certificado inválido para verificar se há conexão em tais casos. A figura abaixo mostra a saída do microcontrolador nas condições descritas.

```
====> WIFI ---> Connecting to NET 52
. 0. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14
====> WiFi connected -----> IP address: 192.168.0.8
Success to open cert file
cert not loaded
connecting to 35.231.29.66
certificate doesn't match
```

**Figura 6. Fingerprint alterado**

Para o último teste realizado, o caminho para o certificado foi trocado, nesse caso, o cliente também não deverá se conectar ao servidor. A figura a seguir mostra a saída do microcontrolador nessas condições.

```
====> WIFI ---> Connecting to NET 52
. 0. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14.
====> WiFi connected -----> IP address: 192.168.0.8
Failed to open cert file
```

**Figura 7. Certificado Ausente**

## **4. Conclusão**

O trabalho aqui discutido é frequentemente atualizado, novas tecnologias e formas de ataque surgem de tempos em tempos. Em relação ao que foi proposto no trabalho, com a tecnologia atual, mostrou um resultado satisfatório no teste proposto, provando sua eficácia.

Em relação ao ambiente proposto, os sensores representaram o ambiente de forma objetiva e precisa, dando credibilidade ao objetivo proposto para solucionar o problema.

No que diz respeito ao objetivo de enviar os dados de forma segura, pode-se verificar que o protocolo TLS garante a segurança dos dados trafegando pela rede, provendo integridade e confidencialidade aos dados entre os pontos.

Sobre o monitoramento em si, o tempo em que o ambiente foi monitorado garantiu ao projeto consistência e providência de dados reais para a análise dos resultados, sendo satisfatório para o que se propõe.

Os resultados do trabalho mostram-se conclusivos, tanto do ponto de vista do monitoramento e visualização dos dados, quanto do ponto de vista da segurança implementada.

Dentre as possibilidades de trabalho que podem ser realizadas a partir deste, tanto em Internet das Coisas quanto em segurança da informação, é possível realizar estudos de escalabilidade do ambiente, adicionando diversos microcontroladores a pulverizadores diferentes, analisando mais de um grupo de sensores de uma só vez. Também é possível abranger outros locais para estudo, testando a eficácia do sistema em diferentes regiões. Outra



possível extensão do ambiente seria a adição de controladores de fluxo no dispositivo, como por exemplo, a ação de aumento ou diminuição na vazão de fungicida remotamente, utilizando sensores específicos no ambiente IoT.

## **Referências**

- OASIS. (2014) MQTT Version 3.1.1. Disponível em: <<http://docs.oasisopen.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf>>. Acesso em: 10 Novembro 2018.
- RIDLEY, Daisy. (2017) “4 Ways in Which IoT is Revolutionising Agriculture”, Disponível em: <<https://datafloq.com/read/4-ways-iot-revolutionising-agriculture/4121>>. Acesso em: 14 Novembro 2018.
- STALLINGS, William. (2008) “Criptografia e segurança de redes: princípios e prática”, 4. ed. São Paulo: Pearson Prentice Hall. 492 p. ISBN 9788576051190.
- VON SOLMS, R and VAN NIEKERK, J (2013) “From information security to cyber security”, computers & security, p. 97-102.